Examples for Automating Infrastructure and Device Management

impress
top gear

実用プレイブック&アイデア満載

Ansible クックブック

大嶋 健容／三枝 浩太／宮﨑 啓史／横地 晃 = 著

Ansibleのさまざまなモジュールを
組み合わせた、実用的なレシピを多数掲載

Linux 運用管理／仮想環境管理（VMware）／コンテナのビルド・デプロイ（Docker、Kubernetes）／Software Defined Network（Cisco ACI）運用管理／ネットワーク管理とデバイス設定の自動化（ルータ／スイッチ／ロードバランサ）

インプレス


Examples for Automating Infrastructure and Device Management
impress
top gear
実用プレイブック&アイデア満載
Ansible
クックブック
大嶋 健容／三枝 浩太／宮﨑 啓史／横地 晃 = 著
Ansibleのさまざまなモジュールを
組み合わせた、実用的なレシピを多数掲載
Linux 運用管理／仮想環境管理（VMware）／コンテナのビルド・デプロイ（Docker、
Kubernetes）／Software Defined Network（Cisco ACI）運用管理／
ネットワーク管理とデバイス設定の自動化（ルータ／スイッチ／ロードバランサ）
インプレス

はじめに

この度は本書をお手にとっていただき誠にありがとうございます。本書は、サーバや仮想化基盤、コンテナ、ネットワーク機器に対するさまざまなユースケースに対応した Ansible のプレイブックサンプル集です。

IT システムを取り巻く日々の業務を効率化したい、オペレーションミスを減らしたいという課題をお持ちの方は多いのではないでしょうか。これらの課題に対して業務の見直しや整理、教育などに加え、自動化も解決手段のひとつです。

Ansible は IT 自動化ツールです。サーバや仮想化基盤、コンテナやネットワーク機器などへの作業を自動化できます。シンプル、パワフル、エージェントレスという特徴を持ち、スキル的にも環境的にもはじめやすいツールです。本書執筆時現在も OSS（オープンソースソフトウェア）として活発に開発が進んでいます。インターネット上の技術情報や事例情報も多く、すでにお使いの方も多いのではないでしょうか。また、大変ありがたいことにこれまで数々の Ansible の入門書が出版されてきました。私自身も学習しはじめの頃は入門書にお世話になりました。

本書は入門書の次のステップに適しています。Linux、VMware、コンテナ、ネットワーク機器に対するユースケースごとに、サンプルプレイブックと解説を掲載しています。ご自身が向き合っている課題に Ansible を利用する際、どのモジュールをどう使えばよいかを知り、Ansible をより活用するための後押しをすることを目的としています。本書がみなさまの自動化の取り組みの一助になれば大変嬉しく思います。

2022 年 2 月吉日
横地 晃

本書のターゲット

本書は、Ansible の基礎知識を持った方をターゲットとしています。とくに、自動化したい対象システムが定まっていて、これから自動化に取り組む、または取り組んでいる方です。「入門書を読んで基本的なことは分かったが、自分が実現したいことに対してプレイブックをどう書いてよいか分からない。」このような場合に、手がかりとなるプレイブックを本書から見つけ、必要に応じて読み替えや公式ドキュメントも併せて参照し、ご自身の課題を解決するケースを想定しています。

なお、基礎知識は、書籍「Ansible 実践ガイド第 3 版」にあてはめると「第 1 章 Ansible の概要」「第 2 章 Ansible の基礎」「第 3 章 プレイブックとインベントリ」が主に該当します。もし基礎知識に不安がある場合は「Ansible 実践ガイド第 3 版」など他の情報も併せてご参照ください。

本書の構成

本書は入門書とは異なり、目的に応じて読みたい章から読む逆引きリファレンス的な書籍です。

前述通り本書は基礎知識を持った方向けですが、Ansible の復習をしたい場合は第 1 章をお読みください。Ansible の特徴や基本的な用語や、近年のアップデートとしてコレクションについても説明しています。

第 2 章では、マネージドノードの種類に依存しないプラットフォーム共通の TIPS を紹介しています。具体的には、プレイブックの実行方法の制御や、実行ログのカスタマイズ、フィルタなどです。

第 3 章以降は、マネージドノードの種類別に章を分けて、ユースケースに応じたプレイブックを紹介しています。Linux、VMware、コンテナ（Docker、Kubernetes）、ルータやスイッチ（Cisco IOS）、SDN（Cisco ACI）、ロードバランサ（F5 BIG-IP Local Traffic Manager）を扱っています。自動化したい対象や興味に応じた章をお読みください。また、各章の最初の節では、対象のマネージドノードを Ansible から扱うための基礎知識や、前提とする環境について説明しています。

なお、サンプルのプレイブックで扱うモジュールのパラメータは一例です。完全なパラメータ一覧は、公式ドキュメントの各モジュールの説明ページを参照してください。適宜、リンクも掲載しています。

本書の解説フォーマットと表記

■ 解説フォーマット

本書の内容は、以下のフォーマットで解説されています。

■ 本書の表記

- コマンドラインのプロンプトは、"$"で示されます（例 1）。
- 実行結果の出力を省略している部分は、"...(略)..."で表記します（例 2）。
- 紙面の幅に収まらないコマンドラインでは、行末に “\” を入れ改行しています（例 3）。
- 紙面の幅に収まらないリストでは、行末に"⇒"を入れ、改行していますが、実際の入力では、1 行で入力してください（例 4）。

・（例 1　コマンドライン）

```
$ ansible-playbook -i inventory.ini step.yml --step  ←コマンドライン

PLAY [sv] **********************************************************************
Perform task: TASK: task1 (N)o/(y)es/(c)ontinue: y     # 実行する  ←コメント
```

・（例 2　出力結果の省略）

```
$ ansible-playbook -i inventory.ini file_creates.yml
...（ 略 ）...
TASK [download from www] *******************************************************************
```

・（例 3　コマンドラインの改行）

```
$ ansible-playbook \  ←行末に \ を入れ改行
    -i ./router-switch/inventory.ini ./router-switch/ios_static_routes.yml \
    --check -vvv
```

・（例 4　リストの改行）

```
1:  tasks:
2:    - name: map sample
3:      ansible.builtin.debug:
4:        msg:
5:          - "{{ sample_strings | map('ansible.builtin.regex_replace',⇒  折り返し
6: '\\s*[0-9\\.]+',  '') }}"
```

本書で使用するコードと実行環境

Ansible をインストールするコントロールノード側の環境は以下のとおりです。

- Ansible 4.2.0
- ansible-core 2.11.2
- Red Hat Enterprise Linux 8.4
- Python 3.9.2

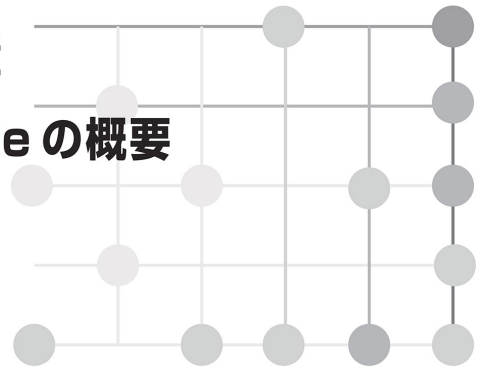
自動化対象であるマネージドノード側の環境は、各章の最初の節にて説明します。

本書で使用するコードは以下の URL で公開しています。ただし、随時修正する可能性がある点をあらかじめご了承ください。

```
https://github.com/ansible-cookbookjp/playbooks
```

目次

第 1 章
Ansible の概要



本章では、基礎知識として Ansible の特徴、モジュールやプレイブックなどの構成コンポーネントについて説明します。Ansible 2.10 からの変化としてコア部分の分離とコレクションの導入があります。コレクションの概要や、インストールなどの操作方法についても紹介します。

1-1 Ansible とは

Ansible は Python 製の IT 自動化ツールです。Linux サーバや Windows サーバのコンフィギュレーション、クラウドリソースの操作、ネットワーク機器の設定変更などのさまざまな作業を自動化できます。自動化したい内容をプレイブックという自動化処理の定義ファイルに記述します。

本書執筆時も OSS（オープンソースソフトウェア）として GitHub のリポジトリ上[*1]で機能追加の開発が進んでいます。リポジトリのフォーク数は 2 万以上にのぼり、開発の活発さが伺えます。現在は、Red Hat, Inc. が「Red Hat Ansible Automation Platform」という製品群としてサポートを提供しています。

1-1-1 特徴

Ansible は、シンプル、パワフル、エージェントレスという 3 つの特徴があります。

■ シンプル

Ansible では YAML というフォーマットを利用して、自動化したい内容を定義します。この定義ファイルをプレイブックと呼びます。YAML はプログラミング言語ではなく、テキストによるデータフォーマットです。そのため、プログラミング言語に慣れていないエンジニアでも学習コストを抑えて始められます。

List 1-1 プレイブック例

```
1: ---
2: - hosts: tokyo
3:
4:   tasks:
5:     - name: start httpd
6:       ansible.builtin.systemd:
7:         name: httpd
8:         state: started
9:       become: true
```

上記のプレイブックの例は、httpd サービスを起動した状態にすることを定義したものです。大まかに何をするものなのか、初見でも容易に想像できるのではないでしょうか。Ansible は適用できる自動

＊1 GitHub リポジトリ ansible/ansible
https://github.com/ansible/ansible

化の範囲が広い分、さまざまな分野のエンジニアが触れる対象になり得ます。そのため、シンプルさや学習コストの低さは重要です。

■ パワフル

Ansible では、モジュールと呼ばれるコンポーネントにより、さまざまな機能が提供されています（Table 1-1）。モジュールを利用することで、詳細な処理を自身で一から作成することなく自動化を実現できます。モジュールは日々の開発を通じて追加、機能改善がされています。

Table 1-1　モジュールの例

モジュール名	概要
ansible.builtin.dnf	dnf によるパッケージのインストール、アップデート、削除
ansible.builtin.systemd	systemd によるサービスの開始、停止、再起動
community.vmware.vmware_guest	VMware vSphere 環境への仮想マシンの作成、削除、各種設定
community.docker.docker_container	Docker コンテナの作成、削除、各種操作
amazon.aws.ec2_vpc_net	AWS 上の VPC を作成、削除
ansible.windows.win_user	Windows のローカルユーザの作成、削除
cisco.ios.ios_config	Cisco IOS のネットワーク機器へのコンフィグ投入
f5networks.f5_modules.bigip_pool	F5 BIG-IP LTM(Local Traffic Manager) のプールの作成、削除
cisco.aci.aci_bd	Cisco ACI(Application Centric Infrastructure) 環境へのブリッジドメイン (BD) の作成、削除、情報取得

また、コマンドをそのまま呼び出すモジュールもあるため、シェルスクリプトなど既存の資産も活用できます。独自のモジュールも開発して利用できるため、提供されているモジュール以外にも機能を追加できる拡張性も備えています。

■ エージェントレス

Ansible は、自動化対象のノードにエージェントをインストールする必要がありません。そのため、自動化を始めやすいという特徴があります。

エージェントが必要なツールの場合は、自動化を始める前にまず自動化対象のノードにエージェントをインストールするという準備が必要です。この準備が導入ハードルを上げてしまいます。

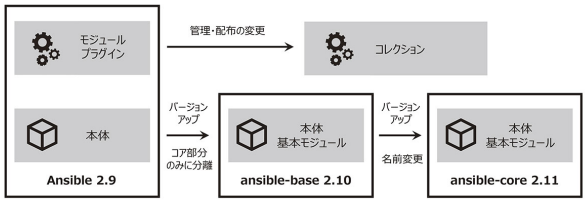
一方、エージェントレス型である Ansible は、自動化対象のノードにこのような準備は不要です。Linux サーバを対象とする場合であれば、手動作業と同じように SSH 接続できて Python が利用できる状態であれば、Ansible から操作できます。このように、Ansible は導入ハードルが低いことが特徴です。

1-1-2 ansible-core と Ansible Community Package

Ansible はバージョン 2.9 までは、Ansible 本体に加えて 3000 を超えるモジュールを内蔵していました。この管理方法はリリースサイクルが遅く、開発された新機能を利用できるようになるまでに時間がかかるなどの課題がありました[*2]。

この課題を解決するために、本体と基本的なモジュールやプラグインをコア部分として残し、その他のモジュールやプラグインはコレクションという管理方式に移行しました。コア部分とコレクションはリリースサイクルが独立しています。そのため、コアのバージョンアップを待たずに、新機能をコレクション経由で素早く利用できるようになりました（Figure 1-1）。コア部分は、Ansible 2.10 では「ansible-base」と名付けられ、2.11 では「ansible-core」に名前が変更されました。

Figure 1-1 コレクションへの移行

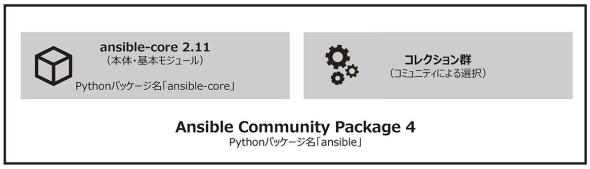


「ansible-core」のみをインストールした場合、必要なコレクションを別途インストールする必要があります。そのため、Ansible 2.9 までのようにさまざまなコレクションを利用するにはインストールの手間がかかってしまいます。そこで便利なのが「Ansible Community Package」です。「Ansible Community Package」は、「ansible-core」とコミュニティが選択したコレクションをセットにしたパッケージです。本書では、この「Ansible Community Package」を利用します（Figure 1-2）。

本書では必要に応じて「Ansible Community Package」または「ansible-core」と表記を区別します。

＊2 Thoughts on Restructuring the Ansible Project
https://www.ansible.com/blog/thoughts-on-restructuring-the-ansible-project

Figure 1-2　ansible-core と Ansible Community Package の関係



1-2　インストール

本書では、Ansible をインストールするノードを**コントロールノード**、自動化対象のノードを**マネージドノード**と呼びます。コントロールノードへのインストールに必要な要件や手順と、マネージドノードの要件を説明します。

1-2-1　コントロールノードの要件

コントロールノードに Ansible をインストールするための要件は以下のとおりです。ただし、Windows はコントロールノードとしてサポートされていないので注意してください。

- Python 2.7 以上、または 3.5 以上

要件の詳細は公式ドキュメント[*3]に記載されています。なお、本書で扱う Ansible 4.2.0 を Python 3.8 未満で実行すると、警告が表示されます。警告の内容は、次期メジャーバージョンのコアである ansible-core 2.12 では、Python3.8 以上でのみサポートとなるというものです。本書では、Red Hat Enterprise Linux 8.4 と Python 3.9 を前提とします。

＊3　Control node requirements
https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html#control-node-requirements

1-2-2　インストール方法

コントロールノードに Ansible をインストールする方法は以下の方法があります。

- Python のパッケージマネージャ（pip）
- 各ディストリビューションのパッケージマネージャ（dnf、yum など）

本書では、Python の仮想環境である venv によって手軽に環境を分けられるメリットがある、pip を利用したインストールを前提とします。

Python3.9 のインストールおよび任意の venv を作成して有効化後に「ansible 4.2.0」をインストールする手順は Operation 1-1 のとおりです。

Operation 1-1　pip による ansible インストール手順

```
$ sudo dnf install python39 -y
$ python3.9 -m venv myvenv
$ source myvenv/bin/activate
$ pip install pip --upgrade
$ pip install ansible==4.2.0
```

これらのコマンドの実行により「ansible」や「ansible-core」、ほかにも Jinja2 などの依存パッケージがインストールされます。インストールできたことを確認するため、バージョンを表示するコマンドを実行します（Operation 1-2）。

Operation 1-2　「ansible --version」によるバージョン確認

```
$ ansible --version
ansible [core 2.11.2]
...(略)...
```

「ansible --version」コマンドで表示されるのは「ansible-core」としてのバージョンです。Operation 1-2 の結果から ansible-core 2.11.2 がインストールされていることが分かります。

「ansible」も「ansible-core」も両方のバージョンを確認する場合は「pip list」コマンドを実行します（Operation 1-3）。

Operation 1-3 「pip list」によるバージョン確認

```
$ pip list
Package       Version
------------- -------
ansible       4.2.0
ansible-core  2.11.2
...(略)...
```

この表示結果により、ansible 4.2.0、ansible-core 2.11.2 がそれぞれインストールされていることが確認できます。

ほかにも、対象となるマネージドノードの種類によっては、追加でパッケージのインストールが必要な場合があります。詳細は各章で説明します。

1-2-3 マネージドノードの要件

自動化対象となるマネージドノード側の準備方法や要件は、その種類よってさまざまです。代表的なユースケースである、Linux サーバをマネージドノードとする場合は以下のとおりです。

- Python 2.6 以上、または Python 3.5 以上
- SSH 接続、ファイル転送方式として SFTP または SCP

この他、仮想化基盤やネットワーク機器などの各プラットフォームの要件については各章で説明します。

1-3 構成コンポーネント

Ansible は、モジュール、インベントリ、プレイブック、プラグイン、ansible.cfg といったコンポーネントで構成されています。

1-3-1 モジュール

モジュールは、特定の処理が作り込まれた機能の単位です。たとえば、サービスの開始、停止を行うモジュールや、パッケージのインストール、アンインストールするモジュールがあります。モジュールに対してパラメータを指定することで処理の詳細を指定し、自動化したい作業を定義します（List 1-2）。

List 1-2　モジュールの利用例

```
1: ---
2: - hosts: tokyo
3:
4:   tasks:
5:     - name: start httpd
6:       ansible.builtin.systemd:  # モジュール名
7:         name: httpd             # パラメータ: 値
8:         state: started          # パラメータ: 値
```

■ 冪等性の考慮

モジュールは冪等性を考慮されて実装されています。冪等性とは同じ操作を何度行っても、結果が変わらない性質です。たとえば、あるパッケージがインストールされた状態にしたい場合、モジュール内部で事前に対象パッケージのインストール状態を確認します。その結果、インストールされていなければインストールし、インストールされていれば何もしません。このため、Ansible の利用者としては、現在の状態を確認して処理を分岐するような考慮は必要ありません。なお、冪等性が考慮されていないモジュールや、使い方によって冪等性が考慮されなくなるモジュールもあります。

1-3-2　インベントリ

インベントリは、自動化対象とするマネージドノードをホストの一覧として定義するものです。INI 形式や YAML 形式で定義します。本書ではシンプルに定義できる INI 形式を利用します。ホストをまとめるグループも定義もできます。グループ化することで、同一グループに所属するホストに同一の処理を適用できるメリットがあります。

■ インベントリ変数

インベントリでは、**ホスト**や**グループ**ごとに利用する**変数**を定義できます。

List 1-3 の例では、tokyo グループに sv01 と sv02 が、osaka グループに sv03 と sv04 が所属することを定義しています。また、各ホストには変数「`ansible_host`」を、グループ tokyo には変数「`region_id`」を定義しています。

List 1-3　INI 形式のインベントリの定義例

```
[tokyo]
sv01 ansible_host=172.16.3.201
sv02 ansible_host=172.16.3.202

[osaka]
sv03 ansible_host=172.16.6.203
sv04 ansible_host=172.16.6.204

; tokyoグループ用の変数
[tokyo:vars]
region_id=3
```

ホスト変数やグループ変数は、インベントリファイルと分離して YAML ファイルとして定義もできます。グループ変数は「group_vars」ディレクトリ配下の「**グループ名**.yml」または「**グループ名/任意のファイル名**.yml」、ホスト変数は「host_vars」ディレクトリ配下の「**ホスト名**.yml」または「**ホスト名/任意のファイル名**.yml」という命名規則に従って配置します（Operation 1-4）。

Operation 1-4　インベントリ変数を変数定義ファイルとして YAML ファイルに分離する例

```
.
├── group_vars
│   ├── osaka.yml    # グループ osaka の変数定義ファイル
│   └── tokyo.yml    # グループ tokyo の変数定義ファイル
└── host_vars
    ├── sv01.yml     # ホスト sv01 の変数定義ファイル
    └── sv03.yml     # ホスト sv03 の変数定義ファイル
```

本書では、公式ドキュメントの「Sample directory layout[*4]」にならって、章ごとにインベントリファイル、「group_vars」ディレクトリ、「host_vars」ディレクトを 1 セット配置する構成を基本とします。

■ インベントリ確認コマンド「ansible-inventory」

定義したインベントリの内容は「ansible-inventory」コマンドで確認できます。「-i」オプションでインベントリを指定します。

「--list」オプションを指定すると、適用される変数を含めてホストの一覧が確認できます（Operation 1-5）。

＊4　Sample directory layout
https://docs.ansible.com/ansible/latest/user_guide/sample_setup.html#sample-directory-layout

Operation 1-5　ansible-inventory コマンドによるホスト一覧の確認

```
$ ansible-inventory -i inventory.ini --list
{
    "_meta": {
        "hostvars": {
            "sv01": {
                "ansible_host": "172.16.3.201",
                "region_id": 3
            },
...(略)...
    "tokyo": {
        "hosts": [
            "sv01",
            "sv02"
        ]
    }
}
```

「--graph」オプションを指定すると、グループとホストの関係を階層構造で確認できます。

Operation 1-6　ansible-inventory コマンドによるホスト階層構造の確認

```
$ ansible-inventory -i inventory.ini --graph
@all:
  |--@osaka:
  |  |--sv03
  |  |--sv04
  |--@tokyo:
  |  |--sv01
  |  |--sv02
  |--@ungrouped:
```

■ インベントリプラグイン

インベントリの仕組みは、後述のプラグインの一種として提供されています。INI 形式であれば「ansible.builtin.ini」**インベントリプラグイン**です。スタティックにホスト情報を定義するタイプだけでなく、「amazon.aws.aws_ec2」のようにクラウドリソースの情報を読み込んで、ダイナミックにインベントリを生成するタイプもあります。インベントリプラグインの一覧は公式ドキュメントを参照してください[*5]。

＊5　Index of all Inventory Plugins
https://docs.ansible.com/ansible/latest/collections/index_inventory.html

1-3-3　プレイブック

プレイブックは、自動化したいマネージドノードへの処理をYAML形式で定義したファイルです。Ansibleで自動化の仕組みを実装していく上で、記述する時間が一番長くなるのがプレイブックです。

プレイブックはいくつかのセクションに分かれています。「hosts」には、操作対象の範囲を指定します。インベントリファイル内で定義したホストやグループの名前を利用します。「tasks」には、処理内容をタスクとして定義します。タスクでは利用するモジュール名を指定し、モジュールに対するパラメータで処理の詳細を指定します。このほか「vars」ではプレイ内で利用する変数を定義できます。

List 1-4　プレイブックの例

```
1: ---
2: - hosts: tokyo    # 対象の指定
3:
4:   # 変数の定義
5:   vars:
6:     service_name: httpd
7:
8:   # タスクの定義
9:   tasks:
10:    - name: start httpd
11:      ansible.builtin.systemd:      # モジュール名の指定
12:        name: "{{ service_name }}"  # パラメータ: 値（変数の利用）
13:        state: started              # パラメータ: 値
14:      become: true
```

基本的には定義したタスクを上から順に処理します。タスクの繰り返し実行や、条件を満たすときのみタスクを実行するといった制御機能も利用できます。

■ プレイブック実行コマンド「ansible-playbook」

プレイブックの実行には「ansible-playbook」コマンドを利用します。「-i」オプションでインベントリを指定し、その後にプレイブックのファイル名を指定します（Operation 1-7）。

Operation 1-7　プレイブックの実行例

```
$ ansible-playbook -i inventory.ini playbook.yml
```

1-3-4 プラグイン

プラグインは、Ansible の機能を補完する仕組みです。プラグインには役割に応じて種別があります。具体的には、マネージドノードへ接続を担うコネクションプラグインや、プレイブックの実行結果の出力を担うコールバックプラグインなどです（Table 1-2）。

Table 1-2 プラグインの種別とプラグインの例

プラグイン種別	プラグイン名	概要
コネクションプラグイン	ansible.builtin.ssh	サーバへ SSH 接続する
コネクションプラグイン	ansible.netcommon.network_cli	ネットワーク機器へ SSH 接続する
ルックアッププラグイン	ansible.builtin.file	ファイルを読み込む
ルックアッププラグイン	ansible.builtin.env	環境変数を読み込む
ルックアッププラグイン	ansible.builtin.template	Jinja2 テンプレートファイルを読み込む
コールバックプラグイン	community.general.yaml	プレイブック実行結果を YAML 形式で表示する
コールバックプラグイン	ansible.posix.profile_tasks	プレイブック実行結果に時刻やタスク実行時間を表示する
インベントリプラグイン	ansible.builtin.ini	INI 形式のインベントリを利用する
インベントリプラグイン	ansible.builtin.yaml	YAML 形式のインベントリを利用する

ヒント プラグインの一覧は公式ドキュメント[*6]を参照してください。

1-3-5 ansible.cfg

`ansible.cfg` は Ansible 自体の動作を指定する設定ファイルです。INI 形式で定義します。設定項目の分類ごとにセクションを分けて指定します。設定項目の一覧は公式ドキュメント[*7]で確認できます。

Table 1-3 ansible.cfg の設定項目例

セクション	設定項目	説明
[defaults]	host_key_checking	マネージドノードへの SSH 接続時に、公開鍵のフィンガープリントをチェックするかどうか。デフォルトは True
[defaults]	interpreter_python	マネージドノード上で実行する Python インタープリタのパス、または自動的に探索するモードの名前を指定する。デフォルトは「`auto_legacy`」モード

＊6 Plugin indexes
`https://docs.ansible.com/ansible/latest/collections/all_plugins.html`
＊7 Ansible Configuration Settings
`https://docs.ansible.com/ansible/latest/reference_appendices/config.html`

List 1-5　ansible.cfg の例

```
1: [defaults]
2: host_key_checking=False
3: interpreter_python=/home/ansible/myvenv/bin/python
```

■ 優先順位

ansible.cfg は複数の場所に配置でき、以下の優先順で検索されます。

（1）環境変数「ANSIBLE_CONFIG」で指定したパスのファイル
（2）カレントディレクトリの「ansible.cfg」
（3）~/.ansible.cfg
（4）/etc/ansible/ansible.cfg

「~/.ansible.cfg」のみ、ファイル名の先頭がドットである点に注意してください。いずれの場所にも ansible.cfg がない場合は、デフォルトの設定で動作します。

■ 設定確認コマンド「ansible-config」

ansible.cfg を読み込んだ結果の設定値は「ansible-config dump」コマンドで確認できます。さらに「--only-changed」オプションを追加すると、デフォルトから変更した設定項目のみに絞り込んで表示できます（Operation 1-8）。

Operation 1-8　デフォルトから変更した設定のみ表示する

```
$ ansible-config dump --only-changed
HOST_KEY_CHECKING(/home/ansible/ansible/ansible.cfg) = False
INTERPRETER_PYTHON(/home/ansible/ansible/ansible.cfg) = /home/ansible/myvenv/bin/python
```

1-4　コレクション

コレクションとはモジュール、プラグイン、プレイブックなどをまとめた管理単位です。コレクションは Ansible 2.8 で試験的な導入が始まりました。その後、Ansible 2.10 でコア部分が切り出され、これまで内蔵していたモジュール群の多くがコレクションに移行しました。これにより、コレクションの本格的な導入が始まりました。

1-4-1　コレクションの概要

コレクション名は「ansible.utils」や「community.general」、「cisco.ios」のようにドット区切りによる名前空間が設けられています。そして、各コレクションの中に含まれるものをさらにドット区切りで続けて表記します。たとえば「cisco.ios」コレクション内の「ios_command」モジュールは「cisco.ios.ios_command」と表記します。この表記を「FQCN(Fully Qualified Collection Name)」と呼びます。なお、ansible-core として標準で内蔵しているものは「ansible.builtin」コレクションとして扱われます[*8]。たとえば systemd モジュールの FQCN 表記は「ansible.builtin.systemd」です。

■ 管理と配布形態

コミュニティとしてのコレクションは Ansible Galaxy[*9]で管理、配布されています。利用したいコレクションを Ansible Galaxy を介してインストールすることで、Ansible の機能を拡張できます。

GitHub のリポジトリもコレクションごとに分かれています。そのため、不具合や最新の開発状況を確認するには、各コレクションのリポジトリを確認します（Figure 1-3）。

Figure 1-3　Ansible Galaxy からリポジトリページに遷移する（ansible.utils コレクションの例）

＊8　ansible.builtin 内のモジュールやプラグインの一覧
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/index.html

＊9　Ansible Galaxy
https://galaxy.ansible.com/

■ Ansible Community Package の同梱コレクション

「1-1-2 ansible-core と Ansible Community Package」で前述したとおり、Ansible Community Package は、コミュニティが選択したコレクションが、同梱コレクションとしてセットでインストールされます。同梱コレクションの一覧は、GitHub 上の「ansible-community/ansible-build-data」というリポジトリ[*10]で管理されています。本書ではとくに補足がない限り Ansible 4.2.0 と、同梱コレクションを利用します。

1-4-2　コレクションの利用

コレクション内のモジュールやプラグインを利用する際は FQCN で指定します。たとえば cisco.ios コレクション内の ios_command モジュールの場合は「cisco.ios.ios_command」と指定します。

List 1-6　プレイブック内で FQCN によるモジュールの指定例

```
1: ---
2: - hosts: rt
3:   gather_facts: false
4:
5:   tasks:
6:     - name: exec show commands
7:       cisco.ios.ios_command:  # FQCN によるモジュールの指定
8:         commands:
9:           - show version
```

■ コレクションの検索対象

FQCN を指定すると設定項目「COLLECTIONS_PATHS」や「COLLECTIONS_SCAN_SYS_PATH」の指定に基づいて、該当のモジュールやプラグインを検索します。

- COLLECTIONS_PATHS[*11]

  コレクションの検索対象パスを指定する設定項目です。デフォルトは「~/.ansible/collections」「/usr/share/ansible/collections」の順です。

＊ 10　ansible 4.2.0 で同梱インストールされるコレクションの一覧
https://github.com/ansible-community/ansible-build-data/blob/main/4/ansible-4.2.0.deps

＊ 11　COLLECTIONS_PATHS
https://docs.ansible.com/ansible/latest/reference_appendices/config.html#collections-paths

- COLLECTIONS_SCAN_SYS_PATH[*12]

  Python のモジュール検索対象パスである sys.path（例: `/home/ansible/myvenv/lib/python3.9/site-packages`）配下を、コレクションの検索対象にするかどうか指定する設定項目です。デフォルトは True です。Ansible Community Package の同梱コレクションは、このパス配下の「`ansible_collections`」ディレクトリにインストールされます。

たとえば `ansible.cfg` で、コレクションの検索対象パスを「`/opt/collections`」に変更する場合は、List 1-7 のように指定します。

List 1-7　ansible.cfg によるコレクション検索対象パスの変更

```
1: [defaults]
2: collections_paths=/opt/collections
```

■ FQCN へのリダイレクト機能

ansible-core は、単にモジュール名のみを指定した場合にどの FQCN にリダイレクトするかの対応表を持っています[*13]。この対応表より、たとえばプレイブック内で「`ios_command`」を指定された場合、FQCN「`cisco.ios.ios_command`」として扱います。ただし、この対応表はあくまで ansible-core 側に定義されているため、各コレクションの開発状況に追従しているわけではありません。そのため、以前の Ansible バージョンで作成されたプレイブックが、FQCN 指定でなくても動作するための互換性確保の仕組みとして扱うのがよいでしょう。

1-4-3　コレクションの操作

コレクションの確認やインストールには「`ansible-galaxy collection`」コマンドを利用します。

■ インストール済みコレクション一覧の確認

インストールされているコレクション一覧を確認するには「`ansible-galaxy collection list`」コマンドを利用します。コレクションがインストールされているディレクトリのパスも併せて表示され

＊ 12　COLLECTIONS_SCAN_SYS_PATH
`https://docs.ansible.com/ansible/latest/reference_appendices/config.html#collections-scan-sys-path`

＊ 13　ansible-core 2.11.2 のリダイレクト対応表「`ansible_builtin_runtime.yml`」
`https://github.com/ansible/ansible/blob/v2.11.2/lib/ansible/config/ansible_builtin_runtime.yml`

ます（Operation 1-9）。

Operation 1-9　コレクション一覧の確認

```
$ ansible-galaxy collection list

# /home/ansible/myenv/lib/python3.9/site-packages/ansible_collections
Collection                    Version
----------------------------- -------
amazon.aws                    1.5.0
ansible.netcommon             2.2.0
ansible.posix                 1.2.0
ansible.utils                 2.3.0
ansible.windows               1.7.0
arista.eos                    2.2.0
awx.awx                       19.2.2
...(略)...
```

設定項目「COLLECTIONS_PATHS」と「COLLECTIONS_SCAN_SYS_PATH」の指定に基づいてコレクションを検索した結果が表示されます。本コマンド実行時のみ一時的にコレクションの検索パスを指定する場合は「-p」オプションでパスを指定します。

■ コレクションのインストール

コレクションをインストールするには「ansible-galaxy collection install コレクション名」コマンドを利用します。インストール対象のコレクション側で、依存コレクションが定義されている場合は、依存コレクションもインストールされます（Operation 1-10）。

Operation 1-10　コレクションのインストール例

```
$ ansible-galaxy collection install cisco.ios
Starting galaxy collection install process
Process install dependency map
...(略)...
cisco.ios:2.3.0 was installed successfully
...(略)...
```

コレクション名と同時に、インストールしたいバージョンも指定できます。たとえば「コレクション名:==2.0.0」で 2.0.0、「’コレクション名:>=2.0.0’」で 2.0.0 以上の最新バージョンをインストールします。「>」のようにシェル上の特別な意味がある文字を含む場合は、クォーテーションで囲う必要があるので注意してください。

また、複数のコレクションを一度にインストールするには、インストールしたいコレクションを定義した「requirements.yml」を利用して、インストール時に読み込む方法が便利です（List 1-8）。

List 1-8　requirements.yml の例

```
---
collections:
  # コレクション名のみの指定
  - ansible.posix

  # バージョンも指定する場合
  - name: ansible.utils
    version: 2.2.0
```

「requirements.yml」を利用してインストールするには「-r」オプションでファイル名を指定します（Operation 1-11）。

Operation 1-11　requirements.yml を利用したインストール

```
$ ansible-galaxy collection install -r requirements.yml
```

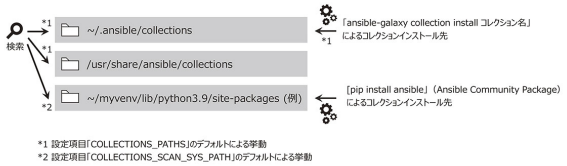
「ansible-galaxy collection install」コマンドによるコレクションのインストール先は、設定項目「COLLECTIONS_PATHS」の指定に基づきます。デフォルトでは「~/.ansible/collections」配下にインストールします。「ansible-galaxy collection install」コマンド実行時のみ、一時的にインストール先を変更する場合は「-p」オプションでパスを指定します（Operation 1-12）。

Operation 1-12　コレクションをインストールしたディレクトリ構造例

```
~/.ansible/collections/
└── ansible_collections
    ├── ansible
    │   ├── netcommon   # このレベルのディレクトリ内にコレクションのコード類がある
    │   ├── posix
    │   └── utils
    └── cisco
        └── ios
```

これまで説明した、コレクションの検索とインストール先についてまとめたものを Figure 1-4 に示します。

Figure 1-4　コレクションの検索とインストール先（デフォルトの場合）



*1 設定項目「COLLECTIONS_PATHS」のデフォルトによる挙動
*2 設定項目「COLLECTIONS_SCAN_SYS_PATH」のデフォルトによる挙動

■ コレクションのアップグレード

インストール済みのコレクションをアップグレードするには「`ansible-galaxy collection install コレクション名 -U`」コマンドを利用します（Operation 1-13）。

Operation 1-13　コレクションのアップグレード例

```
$ ansible-galaxy collection install ansible.utils -U
```

■ コレクションの削除

コレクションを削除する専用のコマンドはありません。そのため、インストール先のディレクトリを削除します。たとえば「`ansible-galaxy collection install ansible.utils`」コマンドで ansible.utils コレクションをインストールした場合、デフォルトでは「`~/.ansible/collections/ansible_collections/ansible/utils`」ディレクトリとして配置されます。このディレクトリを削除することで ansible.utils コレクションを削除できます（Operation 1-14）。

Operation 1-14　コレクションの削除例

```
$ rm -fr ~/.ansible/collections/ansible_collections/ansible/utils
```

Column　Red Hat Developer Program で開発者サブスクリプションを入手

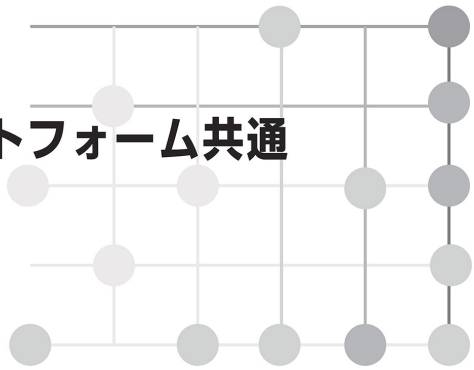
本書では Red Hat Enterprise Linux 8.4 の利用を前提としています。商用利用するには有償のサブスクリプションを購入する必要がありますが、Red Hat Developer Program に登録すると 1 年間の開発者向けサブスクリプションが付与されます。

Red Hat Enterprise Linux や Red Hat Ansible Automation Platform を含むシステムを、16 システムまで利用できます。1 人のユーザに紐付くサブスクリプションのため共用はできませんが、検証、開発用途としては十分ではないでしょうか。

最新情報や製品一覧などの詳細は Web サイトを参照してください。大変有用な制度なので是非登録してみてはいかがでしょうか。

```
https://developers.redhat.com/
```

第２章
プラットフォーム共通



本章以降は、具体的なプレイブックや TIPS を紹介します。まず本章では、各マネージドノードの種類に依存せず、プラットフォーム共通で利用できる内容を扱います。具体的には、プレイブックの実行方法や表示の制御や、フィルタなどを紹介します。

2-1　実行制御

Ansible では、プレイブック内の各タスクを上から順に実行するだけでなく、一時的にタスクを止めたり、特定のタスクのみ実行するなどの制御ができます。このようなタスクの実行を制御する方法を紹介します。

2-1-1　処理を一時的に止める（pause）

キーワード

▷ ansible.builtin.pause モジュール

方　法

プレイの途中で処理を一時的に止めるには ansible.builtin.pause モジュールを利用します（List 2-1）。

List 2-1　ansible.builtin.pause モジュールの利用例: ./common/pause.yml

```
---
- hosts: localhost
  gather_facts: false
  connection: ansible.builtin.local

  tasks:
    - name: pause 10 seconds
      ansible.builtin.pause:
        seconds: 10

    - name: pause 1 minutes
      ansible.builtin.pause:
        minutes: 1
```

解　説

ansible.builtin.pause モジュールの主なパラメータは Table 2-1 のとおりです。

Table 2-1　ansible.builtin.pause モジュールの主なパラメータ

パラメータ名	説明
minutes	停止する分を指定する
seconds	停止する秒数を指定する

停止する秒数もしくは分を定義することで、任意の秒数を停止できます。ポーズ中に Ctrl ＋ C 後に C を入力すると指定した時間を待たずに処理が進みます（Operation 2-1）。

Operation 2-1　プレイブック実行結果例

```
$ ansible-playbook -i inventory.ini pause.yml
PLAY [localhost] ***************************************************************

TASK [pause 10 seconds] ********************************************************
Pausing for 10 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [localhost]

TASK [pause 1 minutes] *********************************************************
Pausing for 60 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [localhost]

PLAY RECAP *********************************************************************
localhost : ok=2  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

また、ポーズ中に Ctrl ＋ C 後に A を入力すると処理を中断できます（Operation 2-2）。

Operation 2-2　ポーズ中に A で中断した例

```
PLAY [localhost] ***************************************************************

TASK [pause 10 seconds] ********************************************************
Pausing for 10 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
Press 'C' to continue the play or 'A' to abort
fatal: [localhost]: FAILED! => {"msg": "user requested abort!"}

NO MORE HOSTS LEFT *************************************************************

PLAY RECAP *********************************************************************
localhost : ok=0  changed=0  unreachable=0  failed=1  skipped=0  rescued=0  ignored=0
```

関　連

▷ ansible.builtin.pause モジュール

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/pause_module.html

2-1-2　changed のときだけ実行するタスクを定義する

キーワード

▷ notify ディレクティブ

▷ handlers

方　法

特定のタスクでマネージドノードに変更を加えたときにのみ必要となる二次的なアクションを定義する方法を紹介します。処理が成功すると必ず changed になる ansible.buitin.command モジュールを利用します（List 2-2）。

List 2-2　handlers の利用例: ./common/handlers.yml

```
---
- hosts: localhost
  gather_facts: false
  connection: ansible.builtin.local

  tasks:
    - name: send command
      ansible.builtin.command: echo "CookBook"
      register: res_command
      notify:
        - handler1
        - after debug2

  handlers:
    - name: after debug1
      ansible.builtin.debug:
        msg: "{{ res_command['stdout_lines'] }}"
      listen: "handler1"

    - name: after debug2
      ansible.builtin.debug:
        msg: "handler2"
```

解　説

changed のときだけ実行するタスクを定義するには、handlers と notify ディレクティブを利用します。changed のときだけ実行するタスクを handlers で定義し、タスクに設定する notify ディレクティブで、どのハンドラを呼び出すかリストで定義します。notify ディレクティブでハンドラの呼び

出し先を指定するには、ハンドラで定義している「name」もしくは「listen」の値にする必要があります。そのためハンドラのタスク名は一意にすることを推奨します。仮に同じタスク名で複数のハンドラが存在する場合、最初に定義されたハンドラが実行されます。複数のタスクから同じ notify が定義されている場合は、複数のタスクの実行ステータスが changed であっても handlers で定義されたタスクが実行されるのは 1 回のみです。また、ハンドラで import_role または include_role モジュールは実行できないので注意してください。

Operation 2-3　実行ログ

```
PLAY [localhost] ***********************************************************

TASK [send command] ********************************************************
changed: [localhost]

RUNNING HANDLER [after debug1] *********************************************
ok: [localhost] => {
    "msg": [
        "CookBook"
    ]
}

RUNNING HANDLER [after debug2] *********************************************
ok: [localhost] => {
    "msg": "handler2"
}

PLAY RECAP *****************************************************************
localhost: ok=3 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

関　連

▷ Handlers: running operations on change

https://docs.ansible.com/ansible/latest/user_guide/playbooks_handlers.html

2-1-3　タスクを非同期に実行する

キーワード

▷ async ディレクティブ

▷ poll ディレクティブ

▷ ansible.builtin.async_status モジュール

方　法

Ansible のデフォルトでは実行しているタスクの結果が確認できるまで、次のタスクが実行されない仕様になっています。もし、タスクの処理をバックグラウンドで実行して、後続のタスクを継続した方が効率がよい場合は、タスクを非同期処理にすることで対応可能です（List 2-3）。

List 2-3　タスクを非同期として実行する例: ./common/async.yml

```
---
- hosts: all
  gather_facts: false
  tasks:
    # (1) sleep コマンドを実行して 15 秒待つ処理を非同期で実行
    - name: execute the wait command to wait 15 sec
      ansible.builtin.command: /bin/sleep 15
      poll: 0
      async: 30
      register: task_status

    # (2) 非同期で実行したジョブの情報を表示
    - name: display task_status
      debug:
        msg: "{{ task_status }}"

    # (3) 非同期で実行したコマンドの終了を待つ
    - name: wait until the wait command is finished
      ansible.builtin.async_status:
        jid: "{{ task_status.ansible_job_id }}"
      register: job_result
      until: job_result.finished
      retries: 60
      delay: 10
```

解　説

非同期タスクで使用する主なディレクティブは Table 2-2 のとおりです。

Table 2-2　非同期タスクで使用する主なディレクティブ

ディレクティブ名	説明
async	タスクを非同期として実行したときの最大実行時間を秒で指定する
poll	実行したタスクが完了するまで状態確認をする間隔を秒で指定する。0 の場合はタスクの状態確認をせずに次のタスクを実行する

ansible.builtin.async_status モジュールで使用する主なパラメータは Table 2-3 のとおりです。

Table 2-3　ansible.builtin.async_status モジュールの主なパラメータ

パラメータ名	説明
jid	ジョブまたはタスク識別子を指定する

(1) では sleep コマンドを使用して 15 秒間待つ処理を実行しています。このタスクは非同期として実行し、結果を待たずに後続タスクを実行するため poll を 0 秒に設定しています。また、async は 30 秒に設定しているため、非同期処理として最大 30 秒間有効なタスクで実行します。もし、タスクが 30 秒以上かかってしまった場合はタスクの成功、失敗かの状態確認はできなくなります。よって、async にはタスクが完了すると思われる時間を設定します。

(2) では非同期で実行したタスクの情報を表示します。本来であれば (1) のタスクが完了するまで本タスクは実行されませんが、(1) を非同期で実行しているため (1) の完了を待たずに本タスクを実行できます。

(3) では ansible.builtin.async_status モジュールを利用して (1) の実行結果を確認します。jid には非同期で実行したタスクのジョブ ID を設定することで、非同期に実行したタスクを特定し状態を確認できます。ジョブ ID は async ディレクティブを使用したタスクの結果を register ディレクティブで保持すると ansible_job_id というキーで参照できます。until、retries、delay を利用して非同期タスクが終了すると思われる時間を設定して状態確認を実行します。

補　足

もし、poll を 1 秒以上に設定した場合は非同期で実行したタスクの状態確認を設定した間隔で実行します。たとえば poll を 5 秒で設定し async を 30 秒で設定した場合は、30 秒内に 5 秒間隔でタスクの状態確認を計 6 回実行します。もし、タスクの処理が SSH のタイムアウトを超える場合には有効な手段として使用可能です。プレイブック例では 15 秒待つ処理を実行しますが、この間は他の入力が発生しません。そのため、仮に SSH のタイムアウトが 15 秒未満の場合はタイムアウトが発生してタスクは失敗します。そこで poll を利用して定期的に状態確認の入力を実行することで、タイムアウトを発生させずにタスクを実行できます。注意点として poll を 1 秒以上にした場合はタスクが完了するまで後続のタスクは実行されません。

関　連

▷ Asynchronous actions and polling

https://docs.ansible.com/ansible/latest/user_guide/playbooks_async.html

▷ ansible.builtin.async_status モジュール

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/async_status_module.html

▷ 2-1-4 非同期処理の完了を待つ

2-1-4　非同期処理の完了を待つ

キーワード

▷ until ディレクティブ

方　法

非同期で実行されている処理の完了を待つには、処理完了の検査を行うタスクに until ディレクティブを使用することで、条件を満たすまでタスクを繰り返して実行できます。例として、ヘルスチェックを行う設定になっている Docker コンテナのステータスが「healthy」になるまで待機するプレイブックは List 2-4 のとおりです。

List 2-4　Docker コンテナのヘルスチェックが「healthy」になるまで待機するプレイブック ./common/until.yml

```
---
- hosts: docker
  gather_facts: false

  tasks:
    - name: wait for container "healthy"
      community.docker.docker_container_info:
        name: http_app
      register: httpapp_info_result
      until: httpapp_info_result.container.State.Health.Status == "healthy"
      retries: 10
      delay: 5
```

解　説

community.docker.docker_container_info モジュールは、Docker コンテナの情報を取得するモジュールです[*1]。register ディレクティブを使用し結果を変数に取得します。コンテナのヘルスチェック[*2]は、たとえば起動に時間のかかるプロセスに対して設定することで、コンテナとしての起動状態とは別に、サービスとして起動中・正常・異常であるというヘルス状態をユーザが定義できます。

通常は現在の状態を 1 回取得するだけのタスクに対して、until ディレクティブを使用すると、取得した状態が特定の値になるまで実行を繰り返すという動作にできます。繰り返し実行で用いる主なディレクティブは Table 2-4 のとおりです。

Table 2-4　繰り返し実行で用いる主なディレクティブ

ディレクティブ名	説明
until	繰り返し実行を終了する条件
retries	再実行の最大回数
delay	再実行時のインターバル時間（秒）

前掲のプレイブックでは、取得したコンテナ情報のヘルスチェックの状態が「healthy」になるまで、5 秒間隔で最大 10 回の再実行を試みます。retries ディレクティブは再実行回数なので、合計の実行回数は最大で 11 回になります。11 回目の実行でも条件を満たさない場合、タスクは失敗となります。条件を満たした場合は、その時点でタスクは成功として終了し、後続のタスクへ処理が継続されます。

応　用

async ディレクティブを使用して非同期で実行したタスクの完了を待つ場合、until ディレクティブと ansible.builtin.async_status モジュールを使用します。このモジュールの使用方法については「2-1-3 タスクを非同期に実行する」を参照してください。

補　足

until ディレクティブは任意のタスクで使用でき、主に register ディレクティブで取得したタスクの実行結果などを条件に処理を繰り返すことで、待機処理を実現できます。類似の機能を提供するモジュールとして対象ホストの接続可否やファイルの状態で待機処理を行う ansible.builtin.wait_for モ

＊1　取得できる情報は「docker container inspect」コマンドと同等です。
＊2　コンテナのヘルスチェック
　　https://docs.docker.com/engine/reference/builder/#healthcheck

ジュールがあります。実現したい処理によって使い分けてください。

関　連

▷ 2-1-3 タスクを非同期に実行する

▷ Retrying a task until a condition is met

https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#retrying-a-task-until-a-condition-is-met

▷ ansible.builtin.wait_for - Waits for a condition before continuing — Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/wait_for_module.html

▷ community.docker.docker_container_info - Retrieves facts about docker container — Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/community/docker/docker_container_info_module.html

2-1-5　特定のタグが付加されたタスクだけ実行する

キーワード

▷ tags ディレクティブ

▷ ansible-playbook -t、--tags オプション

▷ ansible-playbook --skip-tags オプション

方　法

プレイブック内のタスクにはタグを付加できます。タグを付加することにより、特定のタグが付加されたタスクのみ実行、または除外するといった制御ができます。

プレイブックのタスクには List 2-5 のように、tags ディレクティブでタグを付加します。1 つまたは複数のタグを付加できます。

List 2-5　tags ディレクティブの利用例: ./common/tags.yml

```
1: ---
2: - hosts: sv
3:   gather_facts: false
4:
5:   tasks:
6:     - name: task1
```

```
 7:      ansible.builtin.debug:
 8:        msg: task1
 9:      tags:
10:        - tag1
11:        - tag2
12:
13:    - name: task2
14:      ansible.builtin.debug:
15:        msg: task2
16:      tags:
17:        - tag2
18:        - tag3
```

実行したいタスクに付加したタグを、ansible-playbook コマンド実行時に「-t」または「--tags」オプションで指定します。ここでは「-t」オプションを利用します。Operation 2-4 の場合、タスク「task1」のみ実行されます。

Operation 2-4　-t オプションによるタグ指定の実行例

```
$ ansible-playbook -i inventory.ini tags.yml -t tag1
```

解　説

ansible-playbook コマンドの「-t」オプションでタグを指定すると、指定したタグが付加されたタスクのみ実行されます（Operation 2-5）。

Operation 2-5　-t オプションによるタグ指定の実行例

```
$ ansible-playbook -i inventory.ini tags.yml -t tag1

PLAY [sv] **********************************************************************

TASK [task1] *******************************************************************
ok: [sv01] => {
    "msg": "task1"
}

PLAY RECAP *********************************************************************
sv01   : ok=1   changed=0   unreachable=0   failed=0   skipped=0   rescued=0   ignored=0
```

「-t」オプションによるタグはオプションを複数回指定するか、タグのカンマ区切りで複数の指定もできます。複数指定した場合、指定したタグのいずれかが付加されたタスクが実行されます。Operation

2-6 の場合、tag1 が付加されたタスクと tag2 が付加されたタスクが実行されます。

Operation 2-6　-t オプションによるカンマ区切りの複数タグの指定例

```
$ ansible-playbook -i inventory.ini tags.yml -t "tag1,tag2"
```

「-t」オプションとは逆に、指定したタグが付加されたタスクを除外して実行する場合は「--skip-tags」オプションを利用します。

応　用

特別な機能を持つタグとして「never」と「always」があります。それぞれの機能は Table 2-5 のとおりです。

Table 2-5　特別な機能を持ったタグ

タグ名	説明
never	常にスキップされる（ただし「-t never」を指定した場合は実行される）
always	常に実行される（ただし「--skip-tags always」を指定した場合は実行されない）

ansible-playbook コマンドの「-t」オプションで指定する特殊な値として「tagged」と「untagged」があります。それぞれの動作は Table 2-6 のとおりです。

Table 2-6　「-t」オプションで指定する特殊な値

値	説明
tagged	何かしらのタグが付加されているタスクを実行する
untagged	タグが付加されていないタスクを実行する

また、実行対象タスクが期待どおり選択されるか確認するには、ansible-playbook コマンドの「--list-tasks」オプションを併用します。Operation 2-7 の例では「-t tag1」を指定した場合に、タスク「task1」が実行されることが分かります。

Operation 2-7　--list-tasks オプションとの併用例

```
$ ansible-playbook -i inventory.ini tags.yml -t tag1 --list-tasks

playbook: tags.yml
```

```
  play #1 (sv): sv      TAGS: []
    tasks:
      task1     TAGS: [tag1, tag2]
```

「--list-tags」オプションを併用すると、実行対象のタスクに付加されているタグを確認できます。Operation 2-8 の例では「-t tag2」を指定した場合に実行されるタスクに tag1、tag2、tag3 というタグが付加されていることが分かります。

Operation 2-8　--list-tags オプションとの併用例

```
$ ansible-playbook -i inventory.ini tags.yml -t tag2 --list-tags

playbook: tags.yml

  play #1 (sv): sv      TAGS: []
      TASK TAGS: [tag1, tag2, tag3]
```

「--list-tasks」や「--list-tags」オプションは「-t」の指定の有無にかかわらず使用できますが、タグの指定を行っている場合はとくに実行対象タスクの確認に活用してみてください。

補　足

ansible.builtin.include_tasks モジュールなどの ansible.builtin.include_* モジュールを利用するタスクへのタグの付加は注意が必要です。ansible.builtin.include_* モジュール自体のタスクにはタグが付加されますが、読み込んだ内部のタスクやロールには付加されません（継承されない）。

関　連

▷ Tags

https://docs.ansible.com/ansible/latest/user_guide/playbooks_tags.html

▷ ansible.builtin.include_role モジュールや ansible.builtin.include_tasks モジュールでタグが継承されないことの説明

https://docs.ansible.com/ansible/latest/user_guide/playbooks_tags.html#tag-inheritance-for-includes-blocks-and-the-apply-keyword

▷ ansible-playbook コマンド -t、 --tags オプション

https://docs.ansible.com/ansible/latest/cli/ansible-playbook.html#cmdoption-ansible-playbook-tags

▷ ansible-playbook コマンド --skip-tags オプション

https://docs.ansible.com/ansible/latest/cli/ansible-playbook.html#cmdoption-ansible-playbook-skip-tags

2-1-6　プレイブック内の特定のタスク以降を実行する

キーワード

▷ ansible-playbook --start-at-task オプション

方　法

デフォルトでは ansible-playbook コマンドはプレイブックに記述してある最初のタスクから順に実行します。プレイブック内の特定のタスクから実行するには、ansible-playbook コマンドの「--start-at-task」オプションを利用します（Operation 2-9）。

Operation 2-9　--start-at-task オプションの書式

```
$ ansible-playbook \
  -i <インベントリファイル名>  <プレイブックファイル名> \
  --start-at-task <タスク名>
```

解　説

「--start-at-task」オプションには、開始したいタスクの name を指定します。Operation 2-10 の場合、task3 という名前のタスクから実行します。

Operation 2-10　--start-at-task オプション付きのプレイブック実行結果例

```
$ ansible-playbook -i inventory.ini start-at-task.yml --start-at-task task3

PLAY [sv] **********************************************************************

TASK [task3] *******************************************************************
ok: [sv01] => {
    "msg": "this is task3"
}

TASK [task4] *******************************************************************
ok: [sv01] => {
    "msg": "this is task4"
}
```

```
PLAY RECAP *********************************************************************
sv01  : ok=2  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

なお、ansible.builtin.include_* モジュール（ansible.builtin.include_tasks モジュールなど）によって、動的に読み込んだ内部のタスク名は指定できません。

関　連

▷ ansible-playbook コマンド --start-at-task オプション
https://docs.ansible.com/ansible/latest/cli/ansible-playbook.html#cmdoption-ansible-playbook-start-at-task
▷ Executing playbooks for troubleshooting(start-at-task)
https://docs.ansible.com/ansible/latest/user_guide/playbooks_startnstep.html#start-at-task
▷ Comparing includes and imports: dynamic and static re-use
https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse.html#comparing-includes-and-imports-dynamic-and-static-re-use

2-1-7　プレイブックのタスクごとにステップ実行する

キーワード

▷ ステップ実行
▷ ansible-playbook --step オプション

方　法

デフォルトでは ansible-playbook コマンドはプレイブック内に定義されているタスクを順次実行します。一方、プレイブック開発時やトラブルシューティング時は、タスクを一つずつステップ実行して、マネージドノードの状態を都度確認しながら進めたいこともあります。このような場合は、ansible-playbook コマンドの「--step」オプションによるステップ実行が便利です（Operation 2-11）。

Operation 2-11　--step オプションの書式

```
$ ansible-playbook -i <インベントリファイル名> <プレイブックファイル名> --step
```

解　説

「--step」オプションを利用してプレイブックを実行すると、各タスクの実行前に Operation 2-12 のようなプロンプトで一時停止します。

Operation 2-12　一時停止のプロンプト例

```
PLAY [sv] **********************************************************************
Perform task: TASK: task1 (N)o/(y)es/(c)ontinue:
```

続いて、実行前に一時停止したタスクを実行するかどうかを「n」「y」「c」のいずれかを入力して指示します（Table 2-7）。

Table 2-7　タスク実行方法の選択肢

選択肢	説明
n	このタスクを実行せず、次のタスクで再度一時停止する（入力せずに Enter キーを押した場合も同様）
y	このタスクを実行し、次のタスクで再度一時停止する
c	このタスクを実行し、後続のタスクも順次実行する

Operation 2-13 は、task1～4 を定義したプレイブックを「--step」オプションを指定して実行した例です。

Operation 2-13　--step オプション付きのプレイブック実行結果例

```
$ ansible-playbook -i inventory.ini step.yml --step

PLAY [sv] **********************************************************************
Perform task: TASK: task1 (N)o/(y)es/(c)ontinue: y     # 実行する

Perform task: TASK: task1 (N)o/(y)es/(c)ontinue: ******************************

TASK [task1] *******************************************************************
ok: [sv01] => {
    "msg": "this is task1"
}
Perform task: TASK: task2 (N)o/(y)es/(c)ontinue: n     # 実行しない

Perform task: TASK: task2 (N)o/(y)es/(c)ontinue: ******************************
Perform task: TASK: task3 (N)o/(y)es/(c)ontinue: c     # 後続のタスク含めて順次実行する

Perform task: TASK: task3 (N)o/(y)es/(c)ontinue: ******************************
```

```
TASK [task3] *******************************************************************
ok: [sv01] => {
    "msg": "this is task3"
}

TASK [task4] *******************************************************************
ok: [sv01] => {
    "msg": "this is task4"
}

PLAY RECAP *********************************************************************
sv01   : ok=3  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

関　連

▷ ansible-playbook コマンド --step オプション

https://docs.ansible.com/ansible/latest/cli/

ansible-playbook.html#cmdoption-ansible-playbook-step

▷ Executing playbooks for troubleshooting（Step mode）

https://docs.ansible.com/ansible/latest/user_guide/

playbooks_startnstep.html#step-mode

2-2　実行ログ

ansible-playbook コマンドを実行した際に出力されるログのカスタマイズ方法を紹介します。

2-2-1　実行ログの表示形式を変える

キーワード

▷ YAML コールバックプライグイン

方　法

プレイブックの実行ログの表示形式を変更するには、「stdout_callback」を設定する必要があります。実行ログの表示形式として指定できる「stdout_callback」は 1 つだけです。デフォルトの JSON 形式で表示しているログから YAML 形式に切り替える方法を紹介します。YAML コールバックプラグインの指定は環境変数や ansible.cfg ファイルなどで設定します。ansible.cfg の設定は List 2-6 のとおりです。

List 2-6　ansible.cfg における YAML コールバックプラグインの設定例

```
1: [defaults]
2: stdout_callback = community.general.yaml
```

解　説

コールバックプラグインの標準出力の設定をデフォルトの ansible.builtin.default から community.general.yaml に変更することで、実行ログの表示形式が YAML 形式に変更されます。JSON 形式では読みづらい場合は YAML 形式に変更することも検討してください。YAML 形式で出力する設定に変更した実行ログとデフォルトの実行ログは、Operation 2-14、Operation 2-15 のとおりです。

Operation 2-14　YAML 形式に変更した実行ログ

```
PLAY [csr] ***************************************************************

TASK [send command] ******************************************************
ok: [csr]

TASK [debug res command] *************************************************
ok: [csr] =>
  msg:
  - - Cisco IOS XE Software, Version 17.03.03
    - Cisco IOS Software [Amsterdam], Virtual XE Software (X86_64_LINUX_IOSD-UNIVERSALK9-M
  ), Version 17.3.3, RELEASE SOFTWARE (fc7)
    - 'Technical Support: http://www.cisco.com/techsupport'
    - Copyright (c) 1986-2021 by Cisco Systems, Inc.
    - Compiled Thu 04-Mar-21 12:49 by mcpre
    - ''
    - ''
...（略）...
  - - 'Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP'
    - '       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area '
    - '       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2'
    - '       E1 - OSPF external type 1, E2 - OSPF external type 2, m - OMP'
    - '       n - NAT, Ni - NAT inside, No - NAT outside, Nd - NAT DIA'
    - '       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2'
    - '       ia - IS-IS inter area, * - candidate default, U - per-user static ...
    - '       H - NHRP, G - NHRP registered, g - NHRP registration summary'
    - '       o - ODR, P - periodic downloaded static route, l - LISP'
    - '       a - application route'
    - '       + - replicated route, % - next hop override, p - overrides from PfR'
    - '       & - replicated local route overrides by connected'
    - ''
    - Gateway of last resort is 172.31.32.1 to network 0.0.0.0
    - ''
```

```
    - S*    0.0.0.0/0 [1/0] via 172.31.32.1, GigabitEthernet1
    - '      172.31.0.0/16 is variably subnetted, 2 subnets, 2 masks'
    - C        172.31.32.0/20 is directly connected, GigabitEthernet1
    - L        172.31.47.149/32 is directly connected, GigabitEthernet1

PLAY RECAP *********************************************************************
csr: ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

Operation 2-15　デフォルトの実行ログ

```
PLAY [csr] *********************************************************************

TASK [send command] ************************************************
ok: [csr]

TASK [debug res command] *******************************************
ok: [csr] => {
    "msg": [
        [
            "Cisco IOS XE Software, Version 17.03.03",
            "Cisco IOS Software [Amsterdam], Virtual XE Software (X86_64_LINUX_IOSD-UNIVER
SALK9-M), Version 17.3.3, RELEASE SOFTWARE (fc7)",
            "Technical Support: http://www.cisco.com/techsupport",
            "Copyright (c) 1986-2021 by Cisco Systems, Inc.",
            "Compiled Thu 04-Mar-21 12:49 by mcpre",
            "",
            "",
... ( 略 ) ...

        [
            "Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP",
            "       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area ",
            "       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2",
            "       E1 - OSPF external type 1, E2 - OSPF external type 2, m - OMP",
            "       n - NAT, Ni - NAT inside, No - NAT outside, Nd - NAT DIA",
            "       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2",
            "       ia - IS-IS inter area, * - candidate default, U - per-user static route",
            "       H - NHRP, G - NHRP registered, g - NHRP registration summary",
            "       o - ODR, P - periodic downloaded static route, l - LISP",
            "       a - application route",
            "       + - replicated route, % - next hop override, p - overrides from PfR",
            "       & - replicated local route overrides by connected",
            "",
            "Gateway of last resort is 172.31.32.1 to network 0.0.0.0",
            "",
            "S*    0.0.0.0/0 [1/0] via 172.31.32.1, GigabitEthernet1",
```

```
            "      172.31.0.0/16 is variably subnetted, 2 subnets, 2 masks",
            "C        172.31.32.0/20 is directly connected, GigabitEthernet1",
            "L        172.31.47.149/32 is directly connected, GigabitEthernet1"
        ]
    ]
}

PLAY RECAP *********************************************************************
csr: ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

関　連

▷ Callback Plugins

https://docs.ansible.com/ansible/latest/plugins/callback.html

▷ community.general.yaml コールバックプラグイン

https://docs.ansible.com/ansible/latest/collections/community/general/yaml_callback.html

▷ ansible.posix.json コールバックプラグイン

https://docs.ansible.com/ansible/latest/collections/ansible/posix/json_callback.html

▷ ansible.builtin.default コールバックプラグイン

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/default_callback.html

2-2-2　タスクごとの実行時間を計測する

キーワード

▷ プロファイラ

▷ ansible.posix.profile_tasks コールバックプラグイン

方　法

Ansible にはタスクごとの処理時間を計測するためのプロファイラが標準で用意されています。プロファイラは ansible.posix.profile_tasks コールバックプラグインを利用します。コールバックプラグインの指定は、環境変数や `ansible.cfg` ファイルなどで設定します。`ansible.cfg` の設定は List 2-7 のとおりです。

List 2-7　ansible.cfg における profile_tasks コールバックプラグインの設定例

```
1: [defaults]
2: callbacks_enabled = ansible.posix.profile_tasks
```

解　説

ansible.posix.profile_tasks コールバックプラグインを使用する設定の ansible-playbook コマンドの実行結果は Operation 2-16 のとおり[*3]です。処理に時間を要したタスクが時間順に出力されます。また、タスクごとに実行時の日時も出力されます。

Operation 2-16　ansible.posix.profile_tasks コールバックプラグイン使用時の実行例

```
$ ansible-playbook -i inventory.ini file_creates.yml

...（ 略 ）...
TASK [download from www] ********************************************************************
Saturday 21 August 2021  18:31:43 +0900 (0:00:01.344)       0:00:07.092 *******
changed: [rhel8-node]
changed: [fedora-node]

TASK [clone git repository] *****************************************************************
Saturday 21 August 2021  18:31:49 +0900 (0:00:06.415)       0:00:13.507 *******
changed: [fedora-node]
changed: [rhel8-node]

PLAY RECAP **********************************************************************************
fedora-node                : ok=8    changed=8    unreachable=0    failed=0    skipped=0
  rescued=0    ignored=0
rhel8-node                 : ok=8    changed=8    unreachable=0    failed=0    skipped=0
  rescued=0    ignored=0

Saturday 21 August 2021  18:33:23 +0900 (0:01:33.594)       0:01:47.101 *******
===============================================================================
clone git repository ------------------------------------------------------------ 93.59s
download from www --------------------------------------------------------------- 6.42s
copy file to target node -------------------------------------------------------- 1.95s
...（ 略 ）...
```

＊3　使用したプレイブックは「3-3 ファイル操作」の節で解説しているものです。

Column　コールバックプラグイン利用のための設定キー名

本項で紹介した「callbacks_enabled」は ansible-core バージョン 2.11 で新しく追加された設定キーです。それまでのバージョンでは「callback_whitelist」を使用しますが、deprecated になっており、バージョン 2.15 で削除される予定です。バージョン 2.11 でも callback_whitelist は使用可能ですが、警告が表示されます。

補　足

エラーなどで処理が途中で停止した場合は、停止するまでに処理されたタスクの処理時間が表示されます。

関　連

▷ ansible.posix.profile_tasks – adds time information to tasks — Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/posix/profile_tasks_callback.html
▷ CALLBACKS_ENABLED
https://docs.ansible.com/ansible/latest/reference_appendices/config.html#callbacks-enabled

2-3　フィルタ

Ansible には、値の抽出や変換をするフィルタが用意されています。これらのフィルタを利用するプレイブックを紹介します。

2-3-1　辞書のリストから特定条件の要素を抽出する

キーワード

▷ selectattr フィルタ

方　法

辞書型データを要素に持つリストから、特定の条件に合う要素を抽出するには selectattr フィルタを利用します。定義した変数やマネージドノードから取得した情報の中から、特定の値のみを表示した

り条件式に利用したりできます。

条件に合う要素を抽出し、実行ログとして表示するプレイブックは List 2-8 のとおりです。

List 2-8　selectattr フィルタの利用例: ./common/selectattr.yml

```
---
- hosts: sv
  gather_facts: false

  vars:
    # 抽出元データ
    branches:
      - id: 1
        name: tokyo
        network: 10.1.0.0/16
      - id: 2
        name: nagoya
        network: 10.2.0.0/16
      - id: 3
        name: osaka
        network: 10.3.0.0/16

  tasks:
    # 抽出結果の表示
    - name: select branch
      ansible.builtin.debug:
        msg: "{{ branches | selectattr('name', '==', 'nagoya') | list }}"
```

解　説

selectattr フィルタの第一引数には、比較元の辞書のキー、第二引数には比較の演算子、第三引数に比較先の値を指定します。演算子には一致の「==」、不一致の「!=」、比較の「>=」「ge」「>」「gt」「<=」「le」「<」「lt」などが利用できます。この例の場合「name の値が nagoya であるリスト要素を抽出する」という指定です。

プレイブック実行結果は Operation 2-17 のとおりです。条件に合ったリスト要素のみが表示されます。

Operation 2-17　プレイブック実行結果

```
...（ 略 ）...
TASK [select branch] ***********************************************************************
ok: [sv01] => {
```

```
    "msg": [
        {
            "id": 2,
            "name": "nagoya",
            "network": "10.2.0.0/16"
        }
    ]
}
...（ 略 ）...
```

関　連

▷ Testing if a list contains a value

https://docs.ansible.com/ansible/latest/user_guide/playbooks_tests.html#testing-if-a-list-contains-a-value

▷ selectattr Jinja2 フィルタ（Jinja2 ドキュメント）

https://jinja.palletsprojects.com/en/3.0.x/templates/#jinja-filters.selectattr

2-3-2　リストの各要素に対してフィルタを実行する

キーワード

▷ map フィルタ

方　法

リストの各要素に対して、任意のフィルタを実行するには map フィルタを使います。

ansible.builtin.regex_replace フィルタで正規表現置換をリストの各要素に行うには、List 2-9 のように map フィルタの引数に使用するフィルタと、ansible.builtin.regex_replace フィルタの引数を指定します。

List 2-9　map フィルタ利用例: ./common/map.yml

```
1: ---
2: - hosts: sv
3:   gather_facts: false
4:   vars:
5:     sample_strings:
6:       - Red Hat Enterprise Linux 8.4
7:       - Red Hat OpenShift Container Platform 4.7
```

```
 8:       - Red Hat Ansible Automation Platform 4
 9:
10:   tasks:
11:     - name: map sample
12:       ansible.builtin.debug:
13:         msg:
14:           - "{{ sample_strings | map('ansible.builtin.regex_replace',⇒
15: '\\s*[0-9\\.]+',  '') }}"
```

また、辞書型データを要素に持つリストから、特定キーのデータのみを抽出することもできます（List 2-10）。

List 2-10　map フィルタの指定キー抽出例: ./common/map.yml

```
 1: vars:
 2:   branches:
 3:     - id: 1
 4:       name: tokyo
 5:       network: 10.1.0.0/16
 6:     - id: 2
 7:       name: nagoya
 8:       network: 10.2.0.0/16
 9:     - id: 3
10:       name: osaka
11:       network: 10.3.0.0/16
12:
13: tasks:
14:   - name: map with attribute sample
15:     ansible.builtin.debug:
16:       msg:
17:         - "{{ branches | map(attribute='network') }}"
```

解　説

map フィルタは Jinja2 テンプレートの機能として提供されているフィルタです。リストなどのイテレータブルな変数の入力に対し、リストの各要素に実行したい別のフィルタ名を引数に指定します。これにより、Python の map() 組み込み関数と同じように一律ですべての要素に同じ処理を実行できます。第一引数に使用するフィルタ名、第二引数以降に使用するフィルタの引数を指定します。

また、属性名の指定によって辞書型のデータを要素に持つリストに対して、指定したキーのデータのみを取得するフィルタとしても機能します。「List 2-10 map フィルタ指定キー抽出例」では、network キーの値のみを抽出します。このタスクを実行すると結果は Operation 2-18 のとおりです。

Operation 2-18　タスク実行結果

```
TASK [map with attribute sample] ******************************************
ok: [sv01] => {
    "msg": [
        [
            "10.1.0.0/16",
            "10.2.0.0/16",
            "10.3.0.0/16"
        ]
    ]
}
```

関　連

▷ map Jinja2 フィルタ

`https://jinja.palletsprojects.com/en/latest/templates/#jinja-filters.map`

2-3-3　パスワードをハッシュ化する

キーワード

▷ ansible.builtin.password_hash フィルタ

方　法

Linux アカウントの設定などでパスワードを sha512 などでハッシュ化するには、ansible.builtin.password_hash フィルタを使用します。

文字列を入力に、`/etc/shadow` ファイルなどで使用できるハッシュ化されたパスワード文字列を生成します（List 2-11）。

List 2-11　ansible.builtin.password_hash フィルタ利用例: ./common/password_hash.yml

```
1: ---
2: - hosts: sv
3:   gather_facts: false
4:   vars:
5:     sample_password: my_password
6:
7:   tasks:
8:     - name: password_hash sample
9:       ansible.builtin.debug:
```

```
10:         msg: "{{ sample_password | ansible.builtin.password_hash }}"
```

解　説

ansible.builtin.password_hash フィルタは、文字列型の値を入力に、指定のハッシュタイプでハッシュ値に変換します。ハッシュタイプ省略時は sha512 です。

SALT 値は省略時にはランダムになるので、入力のパスワードが同一でも生成されるハッシュ値は実行のたびに異なります。

ansible.builtin.password_hash フィルタ利用例のプレイブックを実行すると Operation 2-19 のとおりですが、実行のたびに値は変化します。そのため、この値を使ったパスワード設定などを行うと、タスクの実行ステータスが毎回「changed」となります。

Operation 2-19　タスク実行結果

```
ok: [sv01] => {
    "msg": "$6$KI1qGhIM755iXENE$wSNk9F ...（ 略 ）... 8XdqmjFO6FcU8ewezFpHVzXncq1JHG2OEL21"
}
```

ハッシュタイプと SALT 値を指定するには、フィルタの第一引数にハッシュタイプ、第二引数に SALT 値を指定します（List 2-12）。

List 2-12　ansible.builtin.password_hash フィルタにハッシュタイプと SALT を指定: ./common/password_hash.yml

```
1: - name: password_hash sample with type and salt
2:   ansible.builtin.debug:
3:     msg: "{{ sample_password | ansible.builtin.password_hash('sha256', 'salt/str') }}"
```

このタスクの内容で実行すると、何度実行しても Operation 2-20 のようになります。

Operation 2-20　ハッシュタイプと SALT を指定した ansible.builtin.password_hash フィルタ実行例

```
ok: [sv01] => {
    "msg": "$5$salt/str$NYEReB9mx5hfTHHl.FjghcQzjtedObplZViH8.dObWB"
}
```

ハッシュタイプには「sha512」「sha256」「md5」などが選べます。Linux アカウントの設定には sha512 が標準的に使用されます。

関　連

▷ man crypt(3)

▷ man shadow(5)

▷ Encrypting and checksumming strings and passwords

https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html#encrypting-and-checksumming-strings-and-passwords

▷ How do I generate encrypted passwords for the user module?

https://docs.ansible.com/ansible/latest/reference_appendices/faq.html#how-do-i-generate-encrypted-passwords-for-the-user-module

2-3-4　リストや辞書から特定条件の要素を抽出する（json_query）

キーワード

▷ community.general.json_query フィルタ

▷ jmespath

方　法

構造化データを JSON 形式とみなし、JMESPath を使って値をクエリするには、community.general.json_query フィルタを使用します（List 2-13）。

List 2-13　community.general.json_query フィルタを利用したプレイブック: ./common/json_query.yml

```
---
- hosts: localhost
  gather_facts: false

  vars:
    # 抽出元データ
    branches:
      - id: 1
        name: tokyo
        network: 10.1.0.0/16
      - id: 2
        name: nagoya
        network: 10.2.0.0/16
      - id: 3
        name: osaka
```

```
16:         network: 10.3.0.0/16
17:
18:   tasks:
19:     # 抽出結果の表示
20:     - name: select branch
21:       ansible.builtin.debug:
22:         msg: "{{ branches | community.general.json_query(query_value) | first }}"
23:       vars:
24:         query_value: "[?name==’nagoya’]"
```

解　説

community.general.json_query フィルタの利用には、コントロールノード側に Python ライブラリの「jmespath」が必要です。フィルタ利用前に pip でインストールします。

community.general.json_query フィルタは JSON 形式のデータから要素またはデータサブセットを抽出できるフィルタです。

プレイブックの実行結果は Operation 2-21 のとおりです。

Operation 2-21　フィルタ実行例

```
...（ 略 ）...
ok: [localhost] => {
    "msg": {
        "id": 2,
        "name": "nagoya",
        "network": "10.2.0.0/16"
    }
}
...（ 略 ）...
```

フィルタで利用するクエリは vars ディレクティブを利用し query_value 変数に格納します。クエリ文字列を別変数に指定することで、フィルタ使用時に複雑になりがちなクォーテーションのエスケープ処理を緩和し、可読性を向上できます。フィルタ対象のリストにアクセスする場合は「[]」を指定します。要素の検索条件を付けるには「?」を付け、その後に対象条件式を指定します。演算子には一致「==」、不一致の「!=」、比較の「>=」「>」「<=」「<」が利用できます。被演算子が数字の場合はバッククォート「‘」で囲みます。たとえば、id が 2 のデータサブセットを抽出する場合は

```
        query_value: "[?id==‘2‘]"
```

と書きます。

応　用

検索したい文字を含む（contains）、文字から始まる（starts_with）、文字で終わる（ends_with）などの条件を付けて要素の抽出をすることができます。応用例として変数 branches のリスト内の name に「a」が含まれているデータサブセットのみを抽出する例を示します。さらに、抽出したデータサブセットの要素を、name と network のみになるよう格納し直しています（List 2-14）。

List 2-14　リスト内の name の値に a が含まれるデータサブセットを抽出し、name と network のみを表示するプレイブック: ./common/json_query_advanced.yml

```
1: tasks:
2:     # リスト内の name の値に a が含まれるデータサブセットを抽出し、name と network のみを表示
3:     - name: select branch
4:       ansible.builtin.debug:
5:         msg: "{{ branches | to_json | from_json | ⇒
6: community.general.json_query(query_value) }}"
7:       vars:
8:         query_value: "[?contains(name,'a')].{name: name, network: network}"
```

Operation 2-22　応用フィルタ実行例

```
...（略）...
ok: [localhost] => {
    "msg": [
        {
            "name": "nagoya",
            "network": "10.2.0.0/16"
        },
        {
            "name": "osaka",
            "network": "10.3.0.0/16"
        }
    ]
}
...（略）...
```

community.general.json_query フィルタで contains、starts_with、ends_with などを利用する際に「`from_json | to_json`」を利用することが公式ドキュメントで紹介されています。

このほかにも、さまざまな条件式を指定でき、柔軟に値を抽出できます。詳細は、community.general.json_query フィルタの内部で使用されている JMESPath の公式ドキュメントを参照してください。

関　連

▷ JMESPath

https://jmespath.org/

▷ Using filters to manipulate data

https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters.html

2-3-5　IPアドレスを扱う

キーワード

▷ ansible.netcommon.ipaddr フィルタ

方　法

IP アドレスを扱うには ansible.netcommon.ipaddr フィルタを利用します（List 2-15）。ipaddr フィルタは ansible.netcommon コレクションに含まれています。未インストールの場合「`ansible-galaxy collection install ansible.netcommon`」を実行し ansible.netcommon コレクションをインストールします。本書では、ansible 4.2.0 をインストール時に同梱されるバージョン 2.2.0 を利用します。

List 2-15　ansible.netcommon.ipaddr フィルタを利用したプレイブック: ./common/ipaddr_filter.yml

```
---
- hosts: localhost
  gather_facts: false
  connection: ansible.builtin.local

  tasks:
    - name: filter ipaddr
      ansible.builtin.debug:
        msg:
          - "{{ '192.168.0.1/24' | ansible.netcommon.ipaddr('address') }}"
          - "{{ '192.168.0.1/24' | ansible.netcommon.ipaddr('network') }}"
          - "{{ '192.168.0.1/24' | ansible.netcommon.ipaddr('netmask') }}"
          - "{{ '192.168.0.1/24' | ansible.netcommon.ipaddr('prefix') }}"
          - "{{ '192.168.0.0/24' | ansible.netcommon.ipaddr('size') }}"
          - "{{ '192.168.0.0/24' | ansible.netcommon.ipaddr('next_usable') }}"
          - "{{ '192.168.0.0/24' | ansible.netcommon.ipaddr('last_usable') }}"
          - "{{ '192.168.0.0/24' | ansible.netcommon.ipaddr('wildcard') }}"
          - "{{ '192.168.0.0/24' | ansible.netcommon.ipaddr('network_wildcard') }}"
```

解　説

ansible.netcommon.ipaddr フィルタの利用には、コントロールノード側に Python ライブラリの「netaddr」が必要です。フィルタ利用前に pip でインストールをします。ansible.netcommon.ipaddr フィルタを利用することで、特定のネットワークのワイルドカードを出力することや、ネットワーク内で利用できる最後の IP アドレスの出力など、さまざまなフィルタリングが可能です。フィルタリングの動作はキーワードによって指定します（Table 2-8）。

Table 2-8　ansible.netcommon.ipaddr フィルタで指定できる主なキーワード

キーワード名	説明
address	IP アドレスのみを表示
network	ネットワークアドレスのみを表示
netmask	ネットマスクのみを表示
prefix	プレフィックス長を表示
size	ネットワークの IP アドレス数を表示
next_usable	指定された IP アドレスのネットワーク内の次に使える IP アドレスを表示
last_usable	指定された IP アドレスのネットワーク内の最後に使える IP アドレスを表示
wildcard	ワイルドカードを表示
network_wildcard	ネットワークとワイルドカードを表示

「List 2-15 ansible.netcommon.ipaddr フィルタを利用したプレイブック」の実行結果は Operation 2-23 のとおりです。

Operation 2-23　ansible.netcommon.ipaddr フィルタ実行例

```
...（ 略 ）...
TASK [filter ipaddr] ***********************************************************
*************
ok: [localhost] => {
    "msg": [
        "192.168.0.1",
        "192.168.0.0",
        "255.255.255.0",
        "24",
        "256",
        "192.168.0.1",
        "192.168.0.254",
        "0.0.0.255",
        "192.168.0.0 0.0.0.255"
    ]
```

```
}
...（ 略 ）...
```

関　連

▷ ipaddr filter

https://docs.ansible.com/ansible/latest/user_guide/playbooks_filters_ipaddr.html

2-4　状態確認

ファクトやモジュールによってマネージドノードから取得した値や定義した変数を、期待した値と比較して状態確認をする方法を紹介します。

2-4-1　期待値チェックをする

キーワード

▷ ansible.builtin.assert モジュール

方　法

ある値が期待どおりかチェックするには ansible.builtin.assert モジュールを利用します。変数を利用する前にチェックして、期待どおりでなければ事前にエラーとすることで、影響範囲を狭くできます。

変数の定義有無、文字列、数値の値やリストをチェックするプレイブックは List 2-16 のとおりです。

List 2-16　ansible.builtin.assert モジュールの利用例: ./common/assert.yml

```
1: ---
2: - hosts: sv
3:   gather_facts: false
4:
5:   # チェック対象の変数の定義
6:   vars:
7:     location: Tokyo
8:     year: 2020
9:     state: present
10:
11:  tasks:
12:    # 変数が定義済みであることをチェック
13:    - name: assert variable definition
```

```
14:      ansible.builtin.assert:
15:        that:
16:          - location is defined
17:          - year is defined
18:          - state is defined
19:
20:    # 文字列と数値の値をチェック
21:    - name: assert value
22:      ansible.builtin.assert:
23:        that:
24:          - location == "Tokyo"          # Tokyo であること
25:          - year >= 2020                 # 2020 以上であること
26:          - state in expected_states     # present か absent のいずれかであること
27:      vars:
28:        expected_states:   # 期待する state の一覧
29:          - present
30:          - absent
```

解　説

ansible.builtin.assert モジュールの主なパラメータは Table 2-9 のとおりです。

Table 2-9　ansible.builtin.assert モジュールの主なパラメータ

パラメータ名	説明
that	期待する条件式を指定する。リストで複数指定する場合は AND 扱い
success_msg	条件に一致する場合のカスタムメッセージ
fail_msg	条件に一致しない場合のカスタムメッセージ

プレイブック実行結果例は Operation 2-24 のとおりです。期待する条件を満たし、タスクのステータスが ok となります。

Operation 2-24　プレイブック実行結果例

```
$ ansible-playbook -i inventory.ini assert.yml

PLAY [sv] **********************************************************************

TASK [assert variable definition] **********************************************
ok: [sv01] => {
    "changed": false,
    "msg": "All assertions passed"
}
```

```
TASK [assert value] ************************************************************
ok: [sv01] => {
    "changed": false,
    "msg": "All assertions passed"
}

PLAY RECAP *********************************************************************
sv01   : ok=2  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

期待する条件式を満たさない場合は fail になります（Operation 2-25）。

Operation 2-25　期待する条件を満たさない場合の結果例

```
...（略）...
TASK [assert value] ************************************************************
fatal: [sv01]: FAILED! => {
    "assertion": "location == \"Tokyo\"",
    "changed": false,
    "evaluated_to": false,
    "msg": "Assertion failed"
}
...（略）...
```

関　連

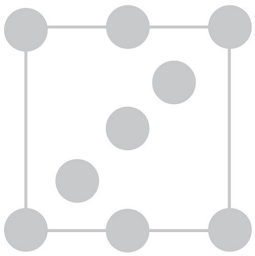
▷ ansible.builtin.assert モジュール

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/assert_module.html

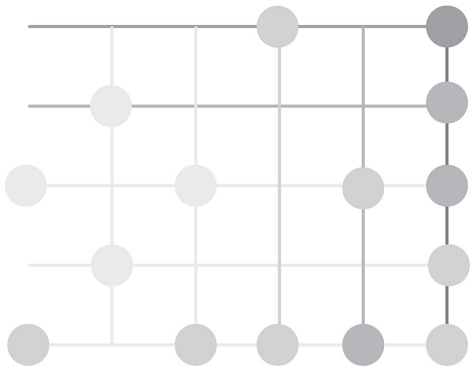
▷ 6-2-5 show コマンド実行結果が期待どおりかチェックする

▷ Tests — Ansible Documentation

https://docs.ansible.com/ansible/latest/user_guide/playbooks_tests.html



第３章
Linux



本章では Linux OS を対象とした Ansible による自動化で使用するモジュールについて紹介します。

Ansible には Linux OS の設定、パッケージソフトとミドルウェアのインストールや設定ファイル変更などを行うためのコレクションとモジュールが豊富に用意されています。

手作業の場合はコンソールや SSH で作業対象サーバへログインし、コマンド実行、ファイル操作、OS やミドルウェアの設定などを実施します。しかし手作業ではオペレーションミスの恐れや、対象サーバが何台もあると、同じ作業を繰り返さなければならず、無用な時間を費やすことになります。

作業の属人化や非効率な作業といった課題は、Ansible を利用し作業を自動化することで解決できます。ミドルウェアや OS の機能を管理する専用のモジュールがあれば、設定する値をモジュールのパラメータとして YAML で記述し、サーバの構築や設定を自動化できます。専用のモジュールがない場合でも、任意のコマンドを実行するモジュールや、ファイル編集を行うモジュールで対応できます。

3-1　Linux 対応の概要

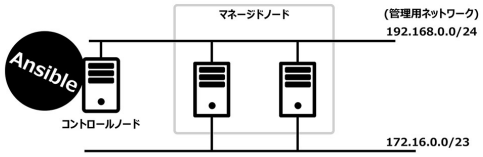
各プレイブックの紹介に先立ち、本章の前提となる環境などについて説明します。

3-1-1　環境

本章において利用するサーバ構成を Figure 3-1 に示します。コントロールノードは RHEL 8.4 を使用しています。マネージドノードも、各プレイブックの紹介でとくに断りがなければ RHEL 8.4 で動作を検証しています。また、一部のタスクについては以下のディストリビューションをマネージドノードとして検証しています。コントロールノードと各マネージドノードは 192.168.0.0/24 の管理用ネットワークで接続されています。172.16.0.0/23 のネットワークについては「3-2-2 ネットワークを設定する」で使用します。

- Fedora 34
- Debian 10.8
- Ubuntu 20.04

Figure 3-1　サーバ構成図



3-1-2　インベントリファイルのグループ

本章で紹介するプレイブックは、Ansible 実行時のインベントリとして、以下のグループを使用しています。それぞれのグループとマネージドノードの Linux ディストリビューションを Table 3-1 に示します。

Table 3-1　インベントリで使用するグループとホスト

グループ名	対象ノード
linux	RHEL、Fedora
linux_apt	Debian、Ubuntu
linux_all	上記すべて

また、「3-4 Web サーバ構築例」の節では、いずれも RHEL 8.4 がマネージドノードですが、Table 3-2 のように環境が異なります。

Table 3-2　Web サーバ構築例の節のグループとホスト

グループ名	説明
private_www	プライベートネットワーク上の RHEL
public_www	AWS 上のグローバル IP アドレスが設定された RHEL

3-1-3　使用コレクションとバージョン

本章で使用するモジュールの ansible.builtin 以外のコレクションとバージョンは Table 3-3 のとおりです。

Table 3-3　本章で使用するコレクションとバージョン一覧

コレクション	バージョン
ansible.posix	1.2.0
community.crypto	1.7.1
community.general	3.3.0

モジュールが依存しているパッケージがある場合は、各モジュールの項、あるいはモジュールドキュメントの Requirements を参照してください。

なお、ファイル編集や OS の基本的な操作については、ansible-core に含まれる ansible.builtin コレクションで提供されます。

3-1-4　接続方式

本章においては、コントロールノードからマネージドノードへの接続は SSH を使用します。認証については公開鍵認証が可能である前提ですが、「3-2-13 SSH 公開鍵設定を行う」で公開鍵設定を Ansible で行う手順についても解説しますので参考にしてください。

特権については必要に応じて become ディレクティブを使用します。本章においては通常ユーザ権限で Ansible を実行し、特権が必要なタスクまたはプレイについては become を使用しています。権限昇格に必要なパスワードは不要な設定になっている前提ですが、パスワード入力を不要にするプレイブックも「3-3-9 lineinfile で文字列を追記または更新する」で解説しているので参考にしてください。

3-2　OS 基本設定

Linux OS におけるユーザ管理やネットワーク設定など OS の基本設定などを行うためのプレイブックを紹介します。

3-2-1　ユーザを作成する

キーワード

▷ useradd コマンド

▷ groupadd コマンド

▷ ansible.builtin.user モジュール

▷ ansible.builtin.group モジュール

方　法

Linux OS にユーザを作成するには、ansible.builtin.user モジュールを使用します。

List 3-1　ansible.builtin.user モジュールの利用例: ./Linux/linux_create_user.yml

```
1: ---
2: - hosts: linux_all
3:   become: true
4:
5:   tasks:
6:     - name: create user
7:       ansible.builtin.user:
8:         name: sample-user
9:         password: "{{ 'sample_password' | ansible.builtin.password_hash('sha512') }}"
10:        uid: 1050
11:        shell: /usr/bin/bash
```

解　説

ansible.builtin.user モジュールで使用する主なパラメータは Table 3-4 のとおりです。

Table 3-4　ansible.builtin.user モジュールの主なパラメータ

パラメータ名	説明
name	ユーザ名
password	ログインパスワードのハッシュ値
uid	ユーザ ID
shell	ログインシェル
home	ホームディレクトリのパス
group	所属グループ
groups	追加の所属グループ

password パラメータは指定した文字列がそのまま/etc/shadow へ書き込まれるため、あらかじめハッシュ化しておく必要があります。ハッシュ化するための password_hash フィルタを使うことで、入力した文字列を/etc/shadow で使える形式の SHA512 ハッシュ値に変換できます。このフィルタについては「2-3-3 パスワードをハッシュ化する」を参照してください。

ホームディレクトリは自動で作成されますが、create_home パラメータに false を指定すると作成されません。

group を指定しない場合、ユーザと同じグループ名とグループ ID が作成され、そのグループが設定されます。グループを明示的に指定したい場合は group パラメータでグループ名を指定します。指定するグループ名が存在しない場合は事前に作成しておく必要があるため、その場合は ansible.builtin.group モジュールを使用してグループを作成してからユーザ作成を行います。

List 3-2　新規作成するグループ指定でユーザ作成: ./Linux/linux_create_user.yml

```
- name: create group
  ansible.builtin.group:
    name: app-group
    gid: 2200

- name: create user with group
  ansible.builtin.user:
    name: app-user
    uid: 1056
    group: app-group
    shell: /sbin/nologin
    home: /opt/app
```

sudo の実行に必要な wheel や sudo グループなど、追加のサブグループ設定を行う場合は group でなく groups にリスト形式で指定します。この groups で指定する追加のサブグループは、append パラ

メータが true の場合は、タスクで指定したサブグループが既存のサブグループに追加されます。一方、false の場合はタスクで指定したサブグループ以外の既存のサブグループ設定は削除されます。append パラメータのデフォルトは true です。

List 3-3　追加グループ指定でユーザ作成: ./Linux/linux_create_user.yml

```
1: - name: create user with additional group
2:   ansible.builtin.user:
3:     name: admin-user
4:     password: "{{ 'admin-password' | ansible.builtin.password_hash('sha512') }}"
5:     shell: /usr/bin/bash
6:     uid: 1250
7:     groups:
8:       - wheel
9:     append: true
```

応　用

state パラメータに absent を指定すると、name パラメータで指定したユーザを削除できます。削除時のデフォルトではユーザが削除されるのみで、ホームディレクトリは残ります。ホームディレクトリなどの関連したディレクトリも削除するには remove パラメータに true を指定します。

関　連

▷ 2-3-3 パスワードをハッシュ化する

▷ ansible.builtin.user – Manage user accounts – Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/ansible/builtin/user_module.html`

▷ ansible.builtin.group – Add or remove groups – Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/ansible/builtin/group_module.html`

3-2-2　ネットワークを設定する

キーワード

▷ NetworkManager

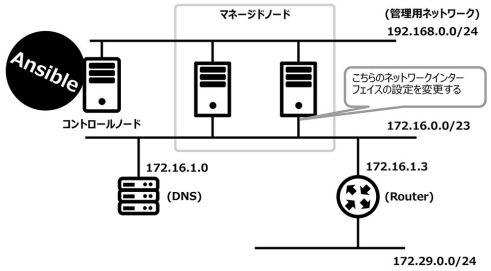
▷ `nmcli` コマンド

▷ community.general.nmcli モジュール

方 法

NetworkManager を使ってネットワークの設定を行っている Linux を対象に、Ansible の community.general.nmcli モジュールを使用してネットワークを設定できます。本項では、Figure 3-2 に示すネットワーク設定における構成図をもとに、マネージドノードの ens224 インターフェイスに、IP アドレス、DNS サーバ、スタティックルートを設定する方法を List 3-4 に示します。

Figure 3-2 ネットワーク設定における構成図



List 3-4 community.general.nmcli モジュールの利用例: ./Linux/linux_nmcli.yml

```
1: ---
2: - hosts: linux
3:   gather_facts: false
4:   become: true
5:
6:   tasks:
7:     - name: configure nic
8:       community.general.nmcli:
9:         type: ethernet
10:        conn_name: ens224
11:        ip4: "{{ ens224_addr }}/{{ ens224_mask }}"
12:        dns4: 172.16.1.0
13:        routes4:
14:          - "172.29.0.0/24 172.16.1.3"
15:        state: present
```

解　説

NetworkManager は、RHEL や Fedora、CentOS などでデフォルトで使用されている Linux のネットワーク設定用のソフトウェアです。Ubuntu など他のディストリビューション[*1]でもインストールすれば使用できます。本項では NetworkManager を使用している環境での自動化について解説します。

Ansible でリモートの Linux サーバに対して自動化を行うには SSH で接続できることが条件です。そのため、Ansible の管理用ネットワークはあらかじめ設定されている必要があります。本モジュールでは、コントロールノードからマネージドノードへの接続設定以外、たとえば DNS 設定やスタティックルートの追加ができます。また、複数の NIC を持つサーバ構成の場合は、管理用ネットワーク以外のネットワークインターフェイスを設定できます。設定できるすべてのパラメータを指定する必要はなく、たとえばスタティックルートの設定のみを追加する場合は、routes4 パラメータのみを指定すれば設定できます。

注意点として、state パラメータに absent を指定した場合は、スタティックルートの設定のみが削除されるのではなく、インターフェイスの定義そのものが削除されるため気を付けてください。

List 3-5　スタティックルートの設定: ./Linux/linux_nmcli.yml

```
1: - name: add static route
2:   community.general.nmcli:
3:     type: ethernet
4:     conn_name: ens224
5:     routes4:
6:       - "172.29.0.0/24 172.16.1.3"
7:     state: present
```

補　足

現バージョンの community.general.nmcli モジュールにはインターフェイスを有効化するためのパラメータがありません。そのため、新規に設定したインターフェイスをアクティブにするには、ansible.builtin.command モジュールを使用して `nmcli` コマンドを実行します。

＊1　Debian のデフォルトのネットワーク設定は`/etc/network/interfaces` です。このファイルは community.general.interfaces_file モジュールを使って操作できます。詳細はドキュメントを参照してください。

List 3-6 nmcli コマンドを使ってインターフェイスの有効化

```
1: - name: update configure
2:   ansible.builtin.command: nmcli con up ens224
```

関 連

▷ NetworkManager

https://networkmanager.dev/

▷ community.general.nmcli – Manage Networking – Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/community/general/nmcli_module.html

3-2-3 パッケージインストールする（Red Hat 系）

キーワード

▷ rpm コマンド

▷ dnf コマンド

▷ yum コマンド

▷ ansible.builtin.dnf モジュール

方 法

RHEL や Fedora などの RPM ベースのディストリビューションを対象に Ansible でパッケージ管理を行うには ansible.builtin.dnf モジュールを使用します。httpd パッケージをインストールするプレイブックは List 3-7 のとおりです。

List 3-7 ansible.builtin.dnf モジュールの利用例: ./Linux/linux_package_install.yml

```
1: - hosts: linux
2:   gather_facts: true
3:   become: true
4:   tasks:
5:     - name: install httpd (dnf)
6:       ansible.builtin.dnf:
7:         name:
8:           - httpd
9:         state: present
```

対象パッケージが 1 つのみの場合は、リスト形式でなく文字列としても指定できます。

List 3-8　対象パッケージが 1 つの場合の ansible.builtin.dnf モジュールの記述例

```
1: - name: install httpd (dnf)
2:   ansible.builtin.dnf:
3:     name: httpd
```

解　説

従来 RPM ベースのパッケージマネージャでは Yum が使用されていました。Ansible モジュールで Yum を実行する場合は ansible.builtin.yum モジュールを使用します。しかし、Yum はサポートが終了した Python2 ベースで実装されているため、Yum に代わる後継のパッケージマネージャとして、DNF がリリースされました。Ansible でも DNF に対応する ansible.builtin.dnf モジュールを使用できます。

ansible.builtin.dnf モジュールの主なパラメータは Table 3-5 のとおりです。

Table 3-5　ansible.builtin.dnf モジュールの主なパラメータ

パラメータ名	説明
name	パッケージ名
state	パッケージの状態

パッケージの状態は Table 3-6 のとおりです。

Table 3-6　パッケージの状態

state パラメータに指定する値	説明
present	インストールする（インストール済みの場合は何もしない）
latest	最新パッケージをインストールする
absent	アンインストールする
installed	present と同じ
removed	absent と同じ

パッケージのバージョンまで指定したい場合は、name パラメータにバージョン番号まで記載します。バージョン番号については、ディストリビューションのサポートページや「`dnf search --showduplicates`」などで確認してください。

List 3-9　バージョン指定した dnf モジュールの利用例: ./Linux/linux_package_install.yml

```
1: - name: install squid (with fixed version)
2:   ansible.builtin.dnf:
3:     name:
4:       - squid-7:4.4-4.*
5:     state: present
```

バージョンは固定せずに、公開されている最新のバージョンのパッケージインストールを行う場合やパッケージのアップデートを行いたい場合は、state パラメータに latest を指定します。latest を指定した場合の動作は以下のとおりです。

- パッケージが未インストールの場合は最新版がインストールされる。
- 最新版でないパッケージがインストール済みの場合は最新版にアップデートされる。
- 最新版のパッケージがインストール済みの場合は変更されない。

present あるいは installed を指定した場合は、「未インストールであればインストールし、インストール済みであればバージョンにかかわらず変更はしない」という動作をします。

インストール済みパッケージをすべて更新したい場合は、name パラメータに「*」と state パラメータに「latest」を指定することで、「dnf -y update」と同じ動作をします。

List 3-10　インストール済みの全パッケージの更新

```
1: - name: update packages
2:   ansible.builtin.dnf:
3:     name:
4:       - '*'
5:     state: latest
```

アップデートで個別の処置が必要なソフトウェアなど、パッケージ更新から除外を行いたい場合は exclude パラメータで指定できます。

List 3-11　特定パッケージを除外した更新

```
1: - name: update packages
2:   ansible.builtin.dnf:
3:     name:
4:       - '*'
5:     state: latest
6:     exclude:
```

```
7:       - kernel*
8:       - kubeadm
```

応　用

パッケージをインストールせずに RPM ファイルをダウンロードすることもできます。download_only パラメータに true を指定し、RPM ファイルをダウンロードするディレクトリを download_dir パラメータに指定します。指定したパッケージの依存関係含めて RPM ファイルがダウンロードされます。

List 3-12　RPM パッケージをダウンロードする

```
1: - name: download packages
2:   ansible.builtin.dnf:
3:     name: ruby
4:     download_only: true
5:     download_dir: /var/tmp/pkg
```

また、パッケージをアンインストールするには、state パラメータに absent を指定します。

補　足

複数のパッケージをインストールする場合はリスト形式でパッケージ名を指定すると 1 回の処理でまとめてインストールされるので効率的です。リスト形式でなくループ処理にすると、ループごとにパッケージ処理が個別に実行されてしまい、処理効率が落ちるので注意してください。詳細は本モジュールのドキュメントの Notes の項を参照してください。

Debian や Ubuntu など、DNF を使ったパッケージ管理ではないディストリビューションに対しては使用できません。後述の「**3-2-4 パッケージインストールする（Debian 系）**」を参照してください。

古いバージョンの RHEL や CentOS のような Yum をパッケージマネージャに使用する Linux の場合は、マネージドノードの OS 上に python2-dnf パッケージを Yum でインストールすれば ansible.builtin.dnf モジュールでパッケージを管理できます。また、Yum 用の ansible.builtin.yum モジュールは Yum だけでなく DNF をパッケージマネージャに使用する Linux にも使用できます。ansible.builtin.yum モジュールは実行時に ansible_pkg_mgr ファクト変数を使ってパッケージマネージャを自動判別する動作がデフォルトです。詳細は ansible.builtin.yum モジュールのドキュメントを参照してください。

関　連

▷ ansible.builtin.dnf – Manages packages with the dnf package manager – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/dnf_module.html
▷ ansible.builtin.yum – Manages packages with the yum package manager – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/yum_module.html

3-2-4　パッケージインストールする（Debian 系）

キーワード

▷ apt コマンド
▷ dpkg コマンド
▷ ansible.builtin.apt モジュール

方　法

Debian や Ubuntu などの、APT をパッケージマネージャに採用しているディストリビューションに対して Ansible でパッケージ管理を行うには、ansible.builtin.apt モジュールを使用します。基本的な使い方は ansible.builtin.dnf モジュールの場合と同じで、パッケージはリスト形式で指定します。

List 3-13　ansible.builtin.apt モジュールの利用例: ./Linux/linux_package_install.yml

```
1: ---
2: - hosts: linux_apt
3:   gather_facts: true
4:   become: true
5:
6:   tasks:
7:     - name: install httpd (apt)
8:       ansible.builtin.apt:
9:         name:
10:          - apache2
11:        state: present
```

解　説

APT は Yum や DNF と異なり、「apt update」コマンドを実行してパッケージインデックスを更新することで公開されている最新のパッケージ情報を取得します。Ansible の ansible.builtin.apt モジュールでは update_cache パラメータの指定でこの操作を行います。

パッケージインデックスの更新のみを行う場合は、update_cache パラメータのみ指定します。

List 3-14　パッケージインデックスの更新

```
1: - name: update package index
2:   ansible.builtin.apt:
3:     update_cache: true
```

インストール済みのパッケージをすべてアップデートするには、upgrade パラメータに true を指定します。この指定により「apt-get upgrade --with-new-pkgs」コマンドと同等の動作になり、すべてのパッケージが更新されます。パッケージインデックスの更新も同時に行う場合は、update_cache パラメータも追加で指定します。

List 3-15　パッケージインデックスの更新でパッケージ更新をまとめて実行

```
1: - name: upgrade all packages
2:   ansible.builtin.apt:
3:     update_cache: true
4:     upgrade: true
```

apt-get upgrade コマンドではなく、apt-get dist-upgrade コマンドと同等の動作が必要な場合は、upgrade のパラメータには dist を指定します。

List 3-16　dist-upgrade の実行

```
1: - name: apt-get dist-upgrade
2:   ansible.builtin.apt:
3:     update_cache: true
4:     upgrade: dist
```

補　足

RHEL や Fedora など、APT を使ったパッケージ管理ではないディストリビューションに対しては使用できません。マネージドノードに RPM 系と APT 系のホストが混在する場合は、gather_facts を有効にしてファクト変数を収集し、「ansible_facts.os_family」や「ansible_facts.pkg_mgr」の値で条件追加してください。

List 3-17　ansible_facts.os_family を使ったパッケージインストールの条件追加例:
./Linux/linux_package_install.yml

```
---
- hosts: linux_all
  gather_facts: true
  become: true

  tasks:
    - name: install httpd (apt)
      ansible.builtin.apt:
        name:
          - apache2
        state: present
      when: ansible_facts.os_family == 'Debian'

    - name: install httpd (dnf)
      ansible.builtin.dnf:
        name:
          - httpd
        state: present
      when: ansible_facts.os_family == 'RedHat'
```

関　連

▷ ansible.builtin.apt – Manages apt-packages – Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/apt_module.html

3-2-5　cron を管理する

キーワード

▷ cron コマンド

▷ crontab コマンド

▷ ansible.builtin.cron モジュール

方　法

Linux でコマンドなどを定期的に実行する cron を Ansible で設定するには ansible.builtin.cron モジュールを使用します。

List 3-18　ansible.builtin.cron モジュールの利用例: ./Linux/linux_cron.yml

```
1: ---
2: - hosts: linux_all
3:   gather_facts: false
4:   become: true
5:
6:   tasks:
7:     - name: cron to user
8:       ansible.builtin.cron:
9:         name: sample_cron_conf
10:        minute: "0"
11:        hour: "*"
12:        job: "date >> /tmp/cron-date-test.log"
```

解　説

Linux にはシステムの cron 設定とユーザごとの cron 設定がありますが、Ansible ではどちらも設定できます。

ユーザごとの cron 設定ファイルはディストリビューションによって多少異なります。RHEL であれば`/var/spool/cron` 以下にユーザ名のファイル名で作成されます。ただし、基本的にはファイルの場所を意識する必要はなく、通常は `crontab -e` コマンドで設定できます。Ansible でユーザごとの cron を設定するには cron_file パラメータを使用せずにタスクを記述します。

user パラメータによるユーザ名の指定は任意です。省略時はマネージドノードへ接続したユーザ、あるいは become 指定があれば昇格したユーザの設定となります。user パラメータを指定した場合は、指定したユーザの cron として設定されます。

List 3-19　ユーザごとの cron 設定: ./Linux/linux_cron.yml

```
1: - name: cron to user
2:   ansible.builtin.cron:
3:     name: sample_cron_conf
4:     minute: "0"
5:     hour: "*"
6:     job: "date >> /tmp/cron-date-test.log"
7:     user: admin-user
```

システムの cron 設定を行う場合は、cron_file パラメータにファイル名を指定します。ここで指定したファイルが`/etc/cron.d` 以下に作成され、対象ノードのシステムワイドな cron 設定として適用されます。この場合はどのユーザの権限で動作するかの指定も必須になるため、user パラメータでユーザ名を指定します。

List 3-20　システムワイドの cron 設定: ./Linux/linux_cron.yml

```
- name: cron conf with file
  ansible.builtin.cron:
    name: cron_conf with cron_file
    minute: "*/15"
    hour: "*"
    job: "date >> /tmp/cron-date-file.log"
    cron_file: ansible_sample
    user: app-user
```

時刻指定の書式など cron 自体の使用方法については crontab(5) などのマニュアル等を参照してください。

補　足

実行にはマネージドノードの OS 上に cron が必要です。DNF や APT で cron または cronie パッケージのインストールが必要です。

関　連

▷ ansible.builtin.cron – Manage cron.d and crontab entries – Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/ansible/builtin/cron_module.html`
▷ man crontab(5)

3-2-6　ファイアウォールを設定する

キーワード

▷ `iptables` コマンド
▷ `firewalld` コマンド
▷ `firewall-cmd` コマンド
▷ ansible.posix.firewalld モジュール

方　法

firewalld の設定を Ansible で行うには、ansible.posix.firewalld モジュールを使用します。

サービス名を指定してアクセス設定を行う場合は、List 3-21 のように記述します。`firewall-cmd` コマンドの`--add-service` オプションに相当します。

List 3-21　サービス名を指定した ansible.posix.firewalld モジュールの利用例: ./Linux/linux_firewalld.yml

```
1:    - name: allow httpd access
2:      ansible.posix.firewalld:
3:        service: http
4:        state: enabled
5:        permanent: true
6:        immediate: true
```

ポート番号を指定した設定の場合は List 3-22 のように記述します。`firewall-cmd` コマンドの `--add-port` オプションに相当します。

List 3-22　ポート番号を指定した ansible.posix.firewalld モジュールの利用例: ./Linux/linux_firewalld.yml

```
1:    - name: allow port number access
2:      ansible.posix.firewalld:
3:        port: 8080/tcp
4:        state: enabled
5:        permanent: true
6:        immediate: true
```

インターフェイスのアクティブゾーンを更新するには List 3-23 のように記述します。List 3-23 に ens224 インターフェイスを trusted ゾーンに割り当てる例を示します。

List 3-23　インターフェイスのゾーンを更新: ./Linux/linux_firewalld.yml

```
1:    - name: ens224 set to trusted zone
2:      ansible.posix.firewalld:
3:        zone: trusted
4:        interface: ens224
5:        state: enabled
6:        permanent: true
7:        immediate: true
```

解　説

ansible.posix.firewalld モジュールで使用する主なパラメータは Table 3-7 のとおりです。

Table 3-7　ansible.posix.firewalld モジュールの主なパラメータ

パラメータ名	説明
interface	対象インターフェイス名
state	通信の許可設定 (enabled:許可、disabled:拒否)

permanent	OS 再起動時に設定を保持するか（true:する、 false:しない）
immediate	設定を即時反映するか（true:する、false:しない）

state パラメータは enabled を指定すると通信を許可し、disabled の場合は拒否します。

permanent パラメータは、永続設定にするかどうかの指定です。firewalld サービスを再起動しても有効にしたい場合は true を指定します。その際、immediate パラメータも true にすることで、タスク実行時に即時反映も行います。permanent パラメータを true に指定して immediate パラメータが false か未設定の場合は、firewalld サービスを再起動するまでは変更は反映されません。permanent パラメータが false の場合は firewalld サービスの再起動後に設定は保持されませんが、immediate パラメータの設定は true とみなされ即時反映されます。

関　連

▷ ansible.posix.firewalld – Manage arbitrary ports/services with firewalld – Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/ansible/posix/firewalld_module.html`

3-2-7　LVM で新しい論理ボリュームを作成する

キーワード

▷ `parted` コマンド
▷ `pvcreate` コマンド
▷ `vgcreate` コマンド
▷ `lvcreate` コマンド
▷ community.general.parted モジュール
▷ community.general.lvg モジュール
▷ community.general.lvol モジュール

方　法

Logical Volume Manager（LVM）で新しい Physical Volume（PV）、Volume Group（VG）および Logical Volume（LV）を作成するには community.general.lvg モジュール、community.general.lvol モジュールを使用します。また、PV を作るために必要なブロックデバイスの操作をするには community.general.parted モジュールを使用します。

List 3-24　LVM を拡張する例: ./Linux/linux_lvm.yml

```
---
- hosts: linux
  gather_facts: true
  tasks:
    # (1) LVM 用のパーティションを作成
    - name: create disk partition for lvm
      community.general.parted:
        device: /dev/sdb
        number: 1
        flags:
          - lvm
        state: present

    # (2) PV と VG の作成
    - name: create volume group
      community.general.lvg:
        vg: example_group
        pvs:
          - /dev/sdb1
        state: present

    # (3) LV の作成
    - name: create logical volume
      community.general.lvol:
        vg: example_group
        lv: example_lv
        size: 100%FREE
```

解　説

community.general.parted モジュールで使用する主なパラメータは Table 3-8 のとおりです。

Table 3-8　community.general.parted モジュールの主なパラメータ

パラメータ名	説明
device	操作対象のブロックデバイスを指定する
number	パーティション番号を指定する
flags	パーティションにセットするフラグを指定する
state	パーティションの状態（present:作成、absent:削除）

community.general.lvg モジュールで使用する主なパラメータは Table 3-9 のとおりです。

Table 3-9　community.general.lvg モジュールの主なパラメータ

パラメータ名	説明
vg	操作対象のボリュームグループ名を指定する
pvs	ボリュームグループで使用する物理デバイスを指定する
state	ボリュームグループの状態（present:作成、absent:削除）

community.general.lvol モジュールで使用する主なパラメータは Table 3-10 のとおりです。

Table 3-10　community.general.lvol モジュールの主なパラメータ

パラメータ名	説明
vg	論理ボリュームで使用するボリュームグループを指定する
lv	操作対象の論理ボリュームを指定する
size	論理ボリュームのサイズを指定する

LV を作成するには必要に応じてブロックデバイスに LVM 用パーティションの作成、PV および VG の操作が必要です。本項で紹介している例ではすべて新規に作成します。

(1) 新規に LVM 用のパーティションをブロックデバイスの/dev/sdb に追加します。パーティションも新規で作成するため番号は 1 にします。
(2) LVM 用で作成した/dev/sdb1 を PV として追加し新規に VG を example_group 名で作成します。
(3) 作成した VG を使用して新規に LV を example_lv という名前で作成します。新規に作成される LV のパスは/dev/example_group/example_lv になります。

関　連

▷ community.general.parted – Configure block device partitions
https://docs.ansible.com/ansible/latest/collections/community/general/parted_module.html
▷ community.general.lvg – Configure LVM volume groups
https://docs.ansible.com/ansible/latest/collections/community/general/lvg_module.html
▷ community.general.lvol – Configure LVM logical volumes
https://docs.ansible.com/ansible/latest/collections/community/general/lvol_module.html

3-2-8　OS を再起動する

キーワード

▷ reboot コマンド

▷ ansible.builtin.reboot モジュール

方　法

OS を再起動するには ansible.builtin.reboot を使用します。

List 3-25　ansible.builtin.reboot の利用例: ./Linux/linux_reboot.yml

```
1: ---
2: - hosts: linux
3:   gather_facts: false
4:   become: true
5:
6:   tasks:
7:     - name: reboot linux
8:       ansible.builtin.reboot:
```

解　説

タイムアウト時間はデフォルトで 600 秒です。ただしこのタイムアウト時間の設定は以下の 2 つがそれぞれ個別にチェックされます。

- reboot コマンドを受け付けてターゲットホストがシャットダウンされるまでで最大 600 秒
- シャットダウンによって SSH が切断されて再び起動して接続できるまでで最大 600 秒

シャットダウンや起動処理に時間がかかるホストなどの場合は、reboot_timeout パラメータでタイムアウト時間を長めに設定します。

List 3-26　タイムアウト時間の設定: ./Linux/linux_reboot.yml

```
1: tasks:
2:   - name: reboot linux
3:     ansible.builtin.reboot:
4:       reboot_timeout: 1200
```

関　連

▷ ansible.builtin.reboot – Reboot a machine – Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/reboot_module.html

3-2-9　SELinux を設定する

キーワード

▷ SELinux

▷ semanage コマンド

▷ ansible.posix.selinux モジュール

方　法

対象ホストにおける SELinux の有効化と無効化の設定を行うには、ansible.posix.selinux を使用します。

List 3-27　ansible.posix.selinux モジュールの利用例: ./Linux/linux_selinux_enable.yml

```
1: - hosts: linux
2:   gather_facts: false
3:   become: true
4:
5:   tasks:
6:     - name: SELinux set to enforcing
7:       ansible.posix.selinux:
8:         policy: targeted
9:         state: permissive
```

以下の解説では、SELinux 自体の有効化・無効化以外にも、ファイルコンテキストやポート設定についても説明します。

解　説

SELinux の有効化・無効化の設定は ansible.posix.selinux モジュールを使用します。state パラメータに指定する値で設定を変更できます（Table 3-11）。

Table 3-11　SELinux の設定状態

state パラメータの値	説明
enforcing	SELinux を有効にする
permissive	ポリシーは強制しないがログ出力する
disabled	SELinux を無効にする

SELinux の設定変更時は OS の再起動が必要ですが、このモジュールは再起動までは行いません。そのため、Ansible で設定変更した場合も別途再起動は必要です。再起動が必要かどうかは、タスク実行の戻り値の reboot_required の真偽値で確認できます。

List 3-28　SELinux の設定変更後再起動する例: ./Linux/linux_selinux_enable.yml

```
- name: SELinux set to enforcing
  ansible.posix.selinux:
    policy: targeted
    state: permissive
  register: selinux_result

- name: reboot os if required
  ansible.builtin.reboot:
  when: selinux_result.reboot_required
```

◇ ファイルコンテキストの設定

SELinux 自体の有効・無効の設定ではなく、ファイルのコンテキスト設定を行うには community.general.sefcontext モジュールを使用します。

/opt/web/以下に対して httpd_sys_content_t を設定するには List 3-29 のように記述します。

List 3-29　ファイルコンテキストの設定: ./Linux/linux_selinux_filecontext.yml

```
- name: SELinux set context
  community.general.sefcontext:
    setype: httpd_sys_content_t
    target: '/opt/web(/.*)?'
```

現在のモジュールのバージョン[*2]ではファイルのコンテキスト変更は反映処理がないため、別途 restorecon を実行する必要があります。

community.general.sefcontext モジュールを使って更新が発生した場合のみ restorecon を ansible.

＊2　community.general コレクションバージョン 3.3.0

builtin.command モジュールを使って実行するには、notify と handlers を使って List 3-30 のように記述します。

List 3-30　notify と handlers を組み合わせて restorecon を行う例: ./Linux/linux_selinux_filecontext.yml

```
1: tasks:
2:   - name: SELinux set context
3:     community.general.sefcontext:
4:       setype: httpd_sys_content_t
5:       target: '/opt/web(/.*)?'
6:     notify:
7:       - restore new SELinux context
8:
9: handlers:
10:   - name: restore new SELinux context
11:     ansible.builtin.command: restorecon -R /opt/web
```

◇ ポートの設定

SELinux のポリシー設定で許可されているポート番号を変更したい場合は community.general.seport モジュールを使用します。

8086/TCP および 8087/TCP で HTTPD によるリッスンを許可するには List 3-31 のように記述します。

List 3-31　SELinux のポートの設定: ./Linux/linux_selinux_port.yml

```
1: - name: allow tcp port 8086,8087 to http
2:   community.general.seport:
3:     ports:
4:       - 8086
5:       - 8087
6:     proto: tcp
7:     setype: http_port_t
```

community.general.sefcontext モジュールと異なり、 community.general.seport モジュールは設定変更が生じた場合は即時反映されます。

補　足

community.general.sefcontext モジュールと community.general.seport モジュールを使うためには policy coreutils-python が必要です。RHEL や Fedora であれば、以下のコマンドを実行することでインストールできます。

```
$ dnf install python3-policycoreutils
```

また、ここで扱った 3 つのモジュールはすべて libselinux-python が必要です。以下のコマンドを実行するとインストールできます。

```
$ dnf install python3-libselinux
```

関　連

▷ 3-2-8 OS を再起動する

▷ ansible.posix.selinux – Change policy and state of SELinux – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/posix/selinux_module.html

▷ community.general.sefcontext – Manages SELinux file context mapping definitions – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/community/general/sefcontext_module.html

▷ community.general.seport – Manages SELinux network port type definitions – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/community/general/seport_module.html

3-2-10　systemd を管理する

キーワード

▷ systemctl コマンド

▷ ansible.builtin.systemd モジュール

方　法

systemctl を使用したサービスの起動設定には、ansible.builtin.systemd モジュールを使用します。

List 3-32　ansible.builtin.systemd モジュールの利用例: ./Linux/linux_service.yml

```
1: ---
2: - hosts: linux
3:   gather_facts: false
4:   become: true
5:
```

```
 6:  tasks:
 7:    - name: enable kubelet
 8:      ansible.builtin.systemd:
 9:        name: kubelet
10:        state: started
11:        daemon_reload: true
12:        enabled: true
13:        masked: false
```

解　説

ansible.builtin.systemd モジュールで使用する主なパラメータは Table 3-12 のとおりです。

Table 3-12　ansible.builtin.systemd モジュールの主なパラメータ

パラメータ名	説明
name	処理対象サービス名
state	設定する状態
daemon_reload	true の場合はユニットファイルを再読み込みする
enabled	true の場合は OS 起動時にサービス起動する
masked	true の場合は起動不可にするマスク設定にする

state パラメータに指定する主な値は Table 3-13 のとおりです。

Table 3-13　ansible.builtin.systemd モジュールの state パラメータに指定する主な値

state パラメータの値	説明
started	起動した状態にする（起動済みの場合は何もしない）
restarted	再起動する（未起動だった場合は起動する）
stopped	停止した状態にする（未起動だった場合は何もしない）

systemd のユニットファイルの変更を行った場合は、コマンドでは「`systemctl daemon-reload`」の実行が必要です。ansible.builtin.systemd モジュールでは daemon_reload パラメータに true を指定します。

また、OS 起動時にサービスを起動したい場合は、enabled パラメータに true を指定します。これで `systemctl enable` 実行時と同様の状態になります。

応　用

`systemctl` コマンドを使った手動の操作も含めてサービスを起動不可とするには、停止かつ無効化にし、さらに masked パラメータでマスク設定も行います。

List 3-33　サービスの停止とマスク設定: ./Linux/linux_service.yml

```
1: - name: disable httpd
2:   ansible.builtin.systemd:
3:     name: httpd
4:     state: stopped
5:     enabled: false
6:     masked: true
```

関　連

▷ ansible.builtin.systemd – Manage systemd units – Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/systemd_module.html

3-2-11　カーネルモジュールをロードする

キーワード

▷ modprobe コマンド

▷ community.general.modprobe モジュール

方　法

カーネルモジュールをロードするには community.general.modprobe モジュールを使用します。

List 3-34　community.general.modprobe モジュールの利用例: ./Linux/linux_kernel_modprobe.yml

```
1: ---
2: - hosts: linux
3:   gather_facts: false
4:   become: true
5:
6:   tasks:
7:     - name: load br_netfilter
8:       community.general.modprobe:
9:         name: br_netfilter
10:        state: present
```

解　説

Linux でカーネルモジュールをロードする場合、コマンドラインでは modprobe コマンドを使用しま

す。Ansible では community.general.modprobe モジュールを使うことで同様の操作ができます。

ただし、CLI ツールの modprobe と同じく一時的にカーネルモジュールをロードする操作になるため、OS を再起動すると設定は消えてしまいます。永続的にモジュールをロードしたい場合は、CLI の場合と同様に「/etc/modules-load.d」ディレクトリ以下にロードするカーネルモジュール名を追加したファイルを作成する必要があります。ファイルの操作には ansible.builtin.copy モジュールや ansible.builtin.template モジュールを利用してください。

応　用

カーネルモジュールをアンロードするには、state に absent を指定します。

関　連

▷ 3-3 ファイル操作

▷ community.general.modprobe – Load or unload kernel modules – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/community/general/modprobe_module.html

3-2-12　カーネルパラメータを設定する

キーワード

▷ sysctl コマンド

▷ ansible.posix.sysctl モジュール

方　法

カーネルのパラメータ設定を行うには、ansible.posix.sysctl モジュールを使用します。

List 3-35　ansible.posix.sysctl モジュールの利用例: ./Linux/linux_kernel_sysctl.yml

```
1: ---
2: - hosts: linux
3:   gather_facts: false
4:   become: true
5:
6:   tasks:
7:     - name: configure sysctl
```

```
 8:      ansible.posix.sysctl:
 9:        name: net.ipv4.ip_forward
10:        value: "0"
11:        state: present
12:        reload: false
13:        sysctl_set: true
14:        sysctl_file: /etc/sysctl.d/k8s.conf
```

解　説

Linux でカーネルパラメータを設定する場合、コマンドラインでは `sysctl` コマンドを使用します。Ansible では ansible.posix.sysctl モジュールで同様の操作ができます。

主要なパラメータは Table 3-14 のとおりです。

Table 3-14　ansible.posix.sysctl モジュールの主なパラメータ

パラメータ名	説明
reload	true の場合は設定ファイル更新と設定反映をセットで行う
sysctl_file	設定を保存するファイルを指定
sysctl_set	true の場合は sysctl -w を使った設定内容のチェックする

reload パラメータのデフォルトは true です。設定ファイル更新と設定反映をセットで行います。false の場合は設定ファイル更新のみで設定反映はされません。

sysctl_file パラメータは設定を書き込むファイルを指定します。未指定時は`/etc/sysctl.conf` が使用されます。

sysctl_set パラメータのデフォルトは false です。false の場合は指定の値を設定ファイルに書き込んでから、reload が true の場合は設定内容の適用が行われるため、パラメータ名や値が誤っている場合はファイル書き込み後にエラーになります。true の場合は設定適用に `sysctl -w` を使用するため、設定ファイルの書き込み前にパラメータ名や値が正しいかチェックされ、不正な内容の場合は設定ファイルへ書き込まれずにエラーになります。

関　連

▷ ansible.posix.sysctl – Manage entries in sysctl.conf. – Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/ansible/posix/sysctl_module.html`

3-2-13　SSH 公開鍵認証を設定する

キーワード

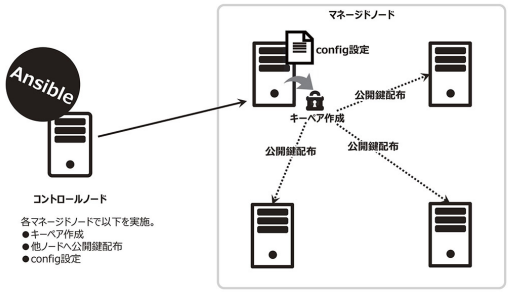
▷ ssh-keygen コマンド
▷ ssh-copy-id コマンド
▷ ssh_config
▷ community.crypto.openssh_keypair モジュール
▷ ansible.posix.authorized_key モジュール
▷ community.general.ssh_config モジュール

方　法

SSH キーペアの作成を行い、公開鍵をリモートホストに登録することで、ノード間の SSH 公開鍵認証設定を行います。SSH のキーペア作成を行う community.crypto.openssh_keypair モジュール、公開鍵をマネージドノードへ登録する ansible.posix.authorized_key モジュール、ssh_config を設定する community.general.ssh_config モジュールを順に使用し、List 3-36 のプレイブックによって全マネージドノード同士で公開鍵認証設定を行います。

処理の概要は Figure 3-3 のとおりです。

Figure 3-3　マネージドノード同士の公開鍵認証設定



このプレイブックは各マネージドノード上の環境変数を参照するため gather_facts を有効にしています。

List 3-36　マネージドノード同士で公開鍵認証設定を行う例: ./Linux/linux_ssh_authorize_nodes.yml

```
---
- hosts: linux
  gather_facts: true

  tasks:
    - name: create directory for ssh keypair
      ansible.builtin.file:
        state: directory
        path: "{{ ansible_env.HOME }}/.ssh/"
        mode: 0700

    - name: create key pair
      community.crypto.openssh_keypair:
        path: "{{ ansible_env.HOME }}/.ssh/id_rsa"
      register: openssh_keypair_result

    - name: authorize public key with managed nodes
      ansible.posix.authorized_key:
        user: "{{ ansible_env.USER }}"
        key: "{{ hostvars[item].openssh_keypair_result.public_key }}"
      loop: "{{ ansible_play_hosts_all }}"

    - name: ssh user config
      community.general.ssh_config:
        host: "{{ item }}"
        user: "{{ hostvars[item].ansible_env.USER }}"
        hostname: "{{ hostvars[item].ansible_facts.default_ipv4.address }}"
        identity_file: "{{ ansible_env.HOME }}/.ssh/id_rsa"
      loop: "{{ ansible_play_hosts_all }}"
```

解　説

◇ SSH キーペアの作成

SSH のキーペアを作成する community.crypto.openssh_keypair モジュールの主なパラメータは Table 3-15 のとおりです。

Table 3-15　community.crypto.openssh_keypair モジュールの主なパラメータ

パラメータ名	説明
path	作成する秘密鍵ファイル名
size	鍵ファイルのサイズ (デフォルトは 4096)
type	作成する鍵の形式 (デフォルトは rsa)

鍵作成時の注意点として、ディレクトリは存在している必要があります。SSH 関連の設定や鍵作成をまったく行っていない状態で`$HOME/.ssh` ディレクトリが存在しない場合に備えて、事前に file モジュールでディレクトリが存在する状態にしておくのが望ましいでしょう。

作成するキーペアのファイル名は、秘密鍵は path で指定したファイル名です。公開鍵は秘密鍵のファイル名に「`.pub`」が付与されたファイル名となります。

◇ SSH 公開鍵の登録

SSH 公開鍵をマネージドノードに登録する ansible.posix.authorized_key モジュールの主なパラメータは Table 3-16 のとおりです。

Table 3-16　ansible.posix.authorized_key モジュールの主なパラメータ

パラメータ名	説明
user	公開鍵を登録するマネージドノード上の OS ユーザ名
key	登録する公開鍵情報

このモジュールでは、user パラメータに指定するユーザ名の「`$HOME/.ssh/authorized_keys`」ファイルが更新されます。

key に指定する公開鍵は、公開鍵ファイルのパスを指定するのではなく、鍵ファイルの内容そのものを指定します。コントロールノードの公開鍵情報をマネージドノードへ登録する場合は、file ルックアッププラグインを使うのが便利です。community.crypto.openssh_keypair モジュールで作成したキーペアを利用する場合は、このモジュールの戻り値の public_key に公開鍵情報がセットされるため、register を使用してキーペア作成の結果を保持して key パラメータに指定できます。

◇ ssh_config の設定

SSH 公開鍵認証の設定ファイルの編集を行う community.general.ssh_config モジュールの主なパラメータは Table 3-17 のとおりです。

Table 3-17　community.general.ssh_config モジュールの主なパラメータ

パラメータ名	説明
user	設定対象のユーザ名
host	接続先ホスト名
identity_file	秘密鍵ファイルのパス
hostname	接続先ホストのアドレス
remote_user	接続先ユーザ名

- user パラメータに指定したユーザの「`$HOME/.ssh/config`」ファイルが処理対象になります。
- host パラメータは ssh_config 内の Host に相当します。接続先ホスト名に使用されます。
- identity_file パラメータは、SSH 公開鍵認証に使用する秘密鍵ファイルのパスを指定します。
- hostname パラメータは実際に接続するアドレスを指定します。ssh_config 内の hostname に相当します。

応　用

鍵認証設定の例として、コントロールノードとマネージドノードで Ansible を使って SSH 公開鍵設定を行う方法について説明します。

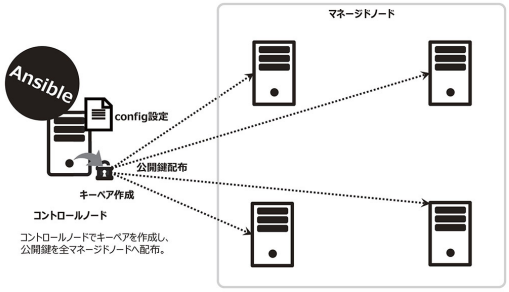
リモートの Linux サーバを対象とした Ansible の利用には、前提条件として SSH 接続可能であることが必要です。パスフレーズなしの SSH 公開鍵認証が設定されていれば認証用パスワードの設定や Ansible 実行時のパスフレーズ等の入力は不要です。そうでない場合はパスワード認証するために ansible_password でパスワードを指定、もしくは `ansible-playbook` コマンド実行時に`-k`（`--ask-pass`）オプションでパスワードを入力する必要があります。本項ではパスワード認証で ansible-playbook を実行、接続して SSH 公開鍵設定を行い、それ以降は公開鍵認証が使用できるようにします。

コントロールノードからマネージドノードへの SSH 公開鍵認証の設定は、以下の流れです。

(1) コントロールノードで SSH キーペアを作成
(2) マネージドノードへ公開鍵を登録
(3) コントロールノードで ssh_config を設定

処理の概要は Figure 3-4 のとおりです。

Figure 3-4　コントロールノードとマネージドノード間の公開鍵認証設定



List 3-37　コントロールノードと各マネージドノード間で公開鍵認証設定を行う例:
./Linux/linux_ssh_authorize_ctrl2target.yml

```
 1:     - name: create directory for ssh keypair on localhost
 2:       ansible.builtin.file:
 3:         state: directory
 4:         path: "{{ lookup('ansible.builtin.env','HOME') }}/.ssh/"
 5:         mode: 0700
 6:       delegate_to: localhost
 7:       run_once: true
 8:
 9:     - name: create key pair
10:       community.crypto.openssh_keypair:
11:         path: "{{ lookup('ansible.builtin.env','HOME') }}/.ssh/id_rsa_ansible"
12:       delegate_to: localhost
13:       run_once: true
14:
15:     - name: authorize public key
16:       ansible.posix.authorized_key:
17:         user: "{{ ansible_env.USER }}"
18:         key: "{{ lookup('ansible.builtin.file', lookup('ansible.builtin.env', ⇒
19: 'HOME') + '/.ssh/id_rsa_ansible.pub' ) }}"
20:
21:     - name: ssh user config
22:       community.general.ssh_config:
23:         user: "{{ ansible_env.USER }}"
```

```
24:         host: "{{ inventory_hostname }}"
25:         hostname: "{{ ansible_facts.default_ipv4.address }}"
26:         identity_file: "{{ lookup('ansible.builtin.env', 'HOME') + ⇒
27: '/.ssh/id_rsa_ansible' }}"
28:       delegate_to: localhost
```

まずキーペアの作成をマネージドノード数にかかわらずコントロールノード上で 1 回だけ実行するために、delegate_to と run_once ディレクティブを併用します。作成するキーペアのパスは Ansible 実行ユーザのホームディレクトリを env ルックアッププラグインで参照して指定します。

次に公開鍵の登録は key パラメータに登録したい公開鍵のファイルの内容を指定します。コントロールノードで作成した公開鍵を指定するには file ルックアッププラグインでファイルのパスを指定して内容を取得し、全マネージドノードへ公開鍵を設定します。

最後にコントロールノードで設定ファイルを更新します。マネージドノードごとに設定するために inventory_hostname マジック変数を使用します。また、全マネージドノードについて設定するため、run_once ディレクティブは使用しません。

補　足

community.general.ssh_config モジュールを使用するには、追加で stormssh パッケージがマネージドノードに必要です。「`pip install stormssh`」でインストールします。

また、パスワード認証を行うにはコントロールノードに sshpass パッケージが必要なため DNF でインストールします。しかし RHEL8 の標準リポジトリではこのパッケージは提供されていません。RHEL の Developer Program であれば利用できる Ansible のリポジトリからインストールするか、Red Hat のサポート外ですが EPEL リポジトリから RHEL8 用の sshpass パッケージを入手してください。

関　連

▷ community.crypto.openssh_keypair – Generate OpenSSH private and public keys – Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/community/crypto/openssh_keypair_module.html`
▷ ansible.posix.authorized_key – Adds or removes an SSH authorized key – Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/ansible/posix/authorized_key_module.html`
▷ community.general.ssh_config – Manage SSH config for user – Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/community/general/`

ssh_config_module.html

▷ Does Red Hat support sshpass? - Red Hat Customer Portal

https://access.redhat.com/solutions/1752733

▷ man ssh_config(5)

3-3　ファイル操作

本節では、ファイルの作成・編集などを行うプレイブックを紹介します。OS 基本設定で紹介したような専用のモジュールがなく、設定ファイルなどを直接変更する必要がある場合などに活用できます。また、Web サーバからのファイルダウンロードや GitHub からリポジトリをクローンするなど、アプリケーションのデプロイの自動化にも応用できます。

3-3-1　ディレクトリを作成する

キーワード

▷ mkdir コマンド

▷ ansible.builtin.file モジュール

方　法

ディレクトリを作成するには ansible.builtin.file モジュールを使用します。

List 3-38　ansible.builtin.file モジュールでディレクトリ作成の例: ./Linux/file_creates.yml

```
---
- hosts: linux
  gather_facts: false
  become: true

  tasks:
    - name: create directory
      ansible.builtin.file:
        path: /var/tmp/ansible-work
        mode: "0755"
        state: directory
        owner: app-user
        group: app-group
```

解　説

Ansible でファイルを作成する操作を行う多くのモジュールは、あらかじめディレクトリが作成されている必要があります。

ディレクトリを作成するためのモジュールは ansible.builtin.file を使用し、state パラメータに directory を指定するとディレクトリを操作します。ディレクトリは再帰的に作成されます。

関　連

▷ ansible.builtin.file – Manage files and file properties – Ansible Documentation

`https://docs.ansible.com/ansible/latest/collections/ansible/builtin/file_module.html`

3-3-2　空ファイルを作成する

キーワード

▷ `touch` コマンド

▷ ansible.builtin.file モジュール

方　法

空ファイルを作成するには、ansible.builtin.file モジュールを使用します。

List 3-39　ansible.builtin.file モジュールで空ファイル作成の例: ./Linux/file_creates.yml

```
---
- hosts: linux
  gather_facts: false
  become: true

  tasks:
    - name: create file
      ansible.builtin.file:
        path: /var/tmp/ansible-work/touch
        mode: "0644"
        state: touch
```

解　説

ansible.builtin.file モジュールの state パラメータに touch を指定することで、Linux の `touch` コマンド相当の処理ができます。

touch コマンドは、指定した名前のファイルが存在しなければ空ファイルを作成し、存在する場合はファイルの更新日時をコマンドの実行時刻あるいは指定の時刻のタイムスタンプに更新するコマンドです。更新日時を実行時刻ではなく、指定した任意の日時にしたい場合は、以下のように、「YYYYmmddHHMM.ss」という書式で指定します。タイムスタンプが更新された場合は、Ansible の実行ステータスは changed になります。

List 3-40　指定日時を設定する場合

```
1: modification_time: "202103311220.01"
2: access_time: "202103311220.01"
```

日時の指定ではなく preserve を指定すると、ファイルが存在していた場合は、タイムスタンプの更新は行わないため、Ansible の実行ステータスは ok になります。preserve を指定してファイルが存在しなかった場合は、デフォルト同様に実行時のタイムスタンプでファイルが作成されます。

List 3-41　ファイル存在時に日付の更新を行わない場合

```
1: modification_time: preserve
2: access_time: preserve
```

関　連

▷ ansible.builtin.file – Manage files and file properties – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/file_module.html

3-3-3　シンボリックリンクを作成する

キーワード

▷ ln コマンド
▷ ansible.builtin.file モジュール

方　法

シンボリックリンクを作成するには、ansible.builtin.file モジュールを使用します。

List 3-42　ansible.builtin.file モジュールでシンボリックリンク作成の例: ./Linux/file_creates.yml

```
---
- hosts: linux
  gather_facts: false
  become: true

  tasks:
    - name: create symlink
      ansible.builtin.file:
        dest: /var/tmp/ansible-work/bin
        src: /usr/local/bin
        state: link
```

解　説

src パラメータで指定したパスのシンボリックリンクを、dest パラメータで指定したパスに作成します。src または dest に指定したパスが存在しない場合は失敗します。

関　連

▷ ansible.builtin.file – Manage files and file properties – Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/file_module.html

3-3-4　マネージドノードへファイルをコピーする

キーワード

▷ cp コマンド

▷ ansible.builtin.copy モジュール

方　法

コントロールノードのファイルシステム上にあるファイルをマネージドノードへコピーするには、ansible.builtin.copy モジュールを使用します。

List 3-43　ansible.builtin.copy モジュールの利用例: ./Linux/file_creates.yml

```
---
- hosts: linux
  gather_facts: false
```

```
4:  become: true
5:
6:  tasks:
7:   - name: copy file to managed node
8:     ansible.builtin.copy:
9:        src: sample.txt
10:       dest: /var/tmp/ansible-work
11:       owner: appuser
12:       group: appgroup
13:       mode: '0664'
```

解　説

ansible.builtin.copy モジュールで使用する主なパラメータは Table 3-18 のとおりです。

Table 3-18　ansible.builtin.copy モジュールの主なパラメータ

パラメータ名	説明
src	コントロールノード上のコピー元ファイルパス
dest	転送先のマネージドノードのファイルパス
owner	ファイルの所有者
group	ファイルのグループ
mode	ファイルのパーミッション
remote_src	true にするとマネージドノード内でのコピー動作

ansible.builtin.copy モジュールを使った転送ファイルデータは、src の代わりに content パラメータにテキストを記述することで、コピー元となるファイルの内容をプレイブックのタスク内に埋め込むこともできます。

content パラメータを使用する場合、YAML の複数行テキストの記述や、Jinja2 テンプレートや変数参照などもできます。

List 3-44　タスク内にインラインでコピーするファイル内容を記述: ./Linux/file_creates.yml

```
1: ---
2: - hosts: linux
3:   gather_facts: false
4:   become: true
5:
6:   tasks:
7:     - name: copy file to managed node
8:       ansible.builtin.copy:
9:         content: |
```

```
10:           hello ansible
11:           {{ lookup('ansible.builtin.env', 'HOSTNAME') }}
12:           {% for host in ansible_play_hosts_all %}
13:           {{ hostvars[host].ansible_host }}
14:           {% endfor %}
15:         dest: /var/tmp/ansible-work/inline_copy.txt
16:         owner: appuser
17:         group: appgroup
18:         mode: '0664'
```

copy モジュールのデフォルト動作はコントロールノードからマネージドノードへのファイル転送です。remote_src パラメータを true に指定することで、マネージドノード上のローカルコピーもできます。List 3-45 のプレイブックの場合、マネージドノード上の src にあるファイルを、同じマネージドノードの dest へコピーします。

List 3-45　マネージドノード内のコピー

```
 1: ---
 2: - hosts: rhel8
 3:   gather_facts: false
 4:   become: true
 5:
 6:   tasks:
 7:     # マネージドノード上のローカルコピー
 8:     - name: copy file to managed node
 9:       ansible.builtin.copy:
10:         src: /var/log/message
11:         dest: /root/log/message
12:         remote_src: true
```

補　足

コピー元のファイルパスを指定する src パラメータとファイルの内容を記述する content は併用できません。

関　連

▷ ansible.builtin.copy – Copy files to remote locations – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/copy_module.html

3-3-5　Jinja2 テンプレートを使用して DNS のゾーンファイルを作成する

キーワード

▷ Jinja2

▷ ansible.builtin.template モジュール

方　法

Jinja2 テンプレートエンジンを使ってファイルを作成し、マネージドノードへ転送するには ansible.builtin.template モジュールを使用します。

DNS のゾーンファイルを生成できる Jinja2 テンプレート（以下、テンプレート）を準備し、ホスト変数や保持している IP アドレスをテンプレートに参照させることによって、ホストグループの情報を持つゾーンファイルを作成できます。

本章では BIND のゾーンファイルを例に扱いますが、テンプレートを `hosts` ファイル向けに修正することで、`hosts` ファイルも作成できます。

List 3-46　ansible.builtin.template モジュールの利用例: ./Linux/file_creates.yml

```
- hosts: dnsserver
  gather_facts: false
  become: true

  tasks:
    - name: create zonefile
      ansible.builtin.template:
        src: templates/named.example.zones_target.j2
        dest: /etc/named/named.example.zones
        group: named
        mode: "0644"
```

List 3-47　ホスト変数を使ったゾーンファイルのテンプレート例: ./Linux/templates/named.example.zones.j2

```
$TTL 3600
@           IN      SOA     ns.example.org. root.example.org.(
                                {{ serial }}      ; Serial
                                3600            ; Refresh
                                900             ; Retry
                                3600000         ; Expire
```

```
 7:                                  3600 )          ; Minimum
 8:             IN      NS      ns.example.org.
 9: ns          IN      A       172.16.0.41
10: {% for host in groups['all'] %}
11: {% if hostvars[host].ens224_addr is defined %}
12: {{ host }}  IN      A       {{ hostvars[host].ens224_addr }}
13: {% endif %}
14: {% endfor %}
```

解　説

DNS 情報として登録したいホスト名と IP アドレスをホスト変数として保持している場合は、これらの情報を使ってゾーンファイルを作成できます。サーバの IP アドレス設定なども含めて Ansible で構築しており、ホスト変数などに必要な変数を保持している場合はこの方法が実装しやすいと思います。

テンプレートでは、指定グループ名のホスト一覧がセットされているマジック変数 groups を使ってループ処理を行っています。ループ内ではホスト変数で保持している IP アドレス情報取得のためにマジック変数 hostvars を使用します。取得した IP アドレスを使ってゾーンファイルを作成します。

インベントリなどでホスト情報を設定していない場合でも、ファクト変数を使ってアドレスを設定することもできます。

List 3-48　ファクト変数を使ったゾーンファイルのテンプレート例: ./Linux/templates/named.example.zones_facts.j2

```
 1: $TTL 3600
 2: @           IN      SOA     ns.example.org. root.example.org.(
 3:                                  {{ serial }}      ; Serial
 4:                                  3600            ; Refresh
 5:                                  900             ; Retry
 6:                                  3600000         ; Expire
 7:                                  3600 )          ; Minimum
 8:             IN      NS      ns.example.org.
 9: ns          IN      A       172.16.0.41
10: {% for host in groups['all'] %}
11: {% if hostvars[host].ansible_facts.ens224 is defined %}
12: {{ host }}  IN      A       {{ hostvars[host].ansible_facts.ens224.ipv4.address }}
13: {% endif %}
14: {% endfor %}
```

関　連

▷ ansible.builtin.template – Template a file out to a target host – Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/

template_module.html

▷ Templating (Jinja2) – Ansible Documentation

https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html

▷ Jinja – Jinja Documentation (3.1.x)

https://jinja.palletsprojects.com/en/latest/

▷ Special Variables – Ansible Documentation

https://docs.ansible.com/ansible/latest/reference_appendices/special_variables.html

3-3-6　Web サーバからファイルをダウンロードする

キーワード

▷ ansible.builtin.get_url モジュール

方　法

指定の URL からファイルをダウンロードするには、ansible.builtin.get_url モジュールを使用します。

Docker Compose（1.29.2）の実行ファイルをダウンロードし、/usr/local/bin/docker-compose へ配置および実行権限を付与する例は List 3-49 のとおりです。

List 3-49　ansible.builtin.get_url モジュールの利用例: ./Linux/file_creates.yml

```
 1: ---
 2: - hosts: linux
 3:   gather_facts: false
 4:   become: true
 5:
 6:   tasks:
 7:     - name: download from www
 8:       ansible.builtin.get_url:
 9:         url: https://github.com/docker/compose/releases/download/1.29.2/⇒
10: docker-compose-Linux-x86_64
11:         dest: /usr/local/bin/docker-compose
12:         owner: root
13:         group: root
14:         mode: '0755'
```

解　説

このモジュールは各マネージドノード上から url パラメータに指定した URL へ HTTP アクセスを行

い、dest パラメータで指定したパスへコンテンツをダウンロードして保存します。

主なパラメータは Table 3-19 のとおりです。

Table 3-19　ansible.builtin.get_url モジュールの主なパラメータ

パラメータ名	説明
url	ファイルをダウンロードする URL
dest	ダウンロードしたファイルを配置するパス
url_username	Basic 認証のユーザ名
url_password	Basic 認証のパスワード
validate_certs	false にすると SSL 証明書検証を省略

関　連

▷ ansible.builtin.get_url – Downloads files from HTTP, HTTPS, or FTP to node – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/get_url_module.html

3-3-7　Git のリモートリポジトリからクローンする

キーワード

▷ git コマンド

▷ ansible.builtin.git モジュール

方　法

Git のリモートリポジトリをクローンするには ansible.builtin.git モジュールを使用します。

ソースコードを管理する GitHub や GitLab などの SaaS のリポジトリ（GitLab の場合はプロジェクト）や、プライベート環境に構築したリポジトリの URL を指定できます。

List 3-50　ansible.builtin.git モジュールの利用例: ./Linux/file_creates.yml

```
1: ---
2: - hosts: linux
3:   gather_facts: false
4:   become: true
5:
6:   tasks:
7:     - name: clone git repository
```

```
 8:     ansible.builtin.git:
 9:       repo: https://github.com/ansible/ansible.git
10:       dest: /usr/local/src/ansible
11:       version: stable-2.11
```

解　説

ansible.builtin.git モジュールで使用する主なパラメータは Table 3-20 のとおりです。

Table 3-20　ansible.builtin.git モジュールの主なパラメータ

パラメータ名	説明
repo	リポジトリの URL
dest	リポジトリを保存するディレクトリパス
version	ブランチ名やタグ名
key_file	SSH 秘密鍵のファイルパス
single_branch	true の場合はシングルブランチをクローン
depth	クローンするコミット数

dest パラメータにはクローン先のローカルリポジトリになるパスを指定します。CLI の「`git clone`」はデフォルトでリポジトリ名のディレクトリが作成されますが、本モジュールでは作成されません。タグやブランチ名を指定するには version パラメータに指定します。

認証が必要なリポジトリの場合で HTTP または HTTPS アクセスを行う場合は、URL のホスト名部分に@記号を付与してユーザ名とパスワードを埋め込むことでクローンできます。ただしその際、Ansible の実行ログ内 URL の一部にパスワード文字列が表示されてしまいます。そのため、no_log ディレクティブに true を設定し、ログが表示されないようにするのが望ましいでしょう。

List 3-51　HTTPS 認証を行う Git クローン

```
1: - name: git clone with authenticated
2:   ansible.builtin.git:
3:     repo: https://username:password@gitlab.example.org:8443/sample/sample-app.git
4:     dest: /var/tmp/repo/sample-app
5:   no_log: true
```

GitHub や GitLab のように、SSH 公開鍵をリポジトリに登録しておくことで SSH を使った Git クローンもできます。

SSH の公開鍵認証に使用する秘密鍵のパスを key_file パラメータで指定します。

List 3-52　SSH 認証を行う Git クローン

```
1: - name: git clone
2:   ansible.builtin.git:
3:     repo: ssh://git@gitlab-ce.example.org:25022/sample/sample-app.git
4:     dest: /var/tmp/repo/sample-app
5:     key_file: /home/user/.ssh/id_rsa_gitlab
```

マネージドノードのユーザの$HOME/.ssh/config で使用する秘密鍵などの設定がされていれば、key_file パラメータは省略できます。

single_branch と depth パラメータについては、Git のコマンドオプションと同様です。詳細については Git のドキュメントを参照してください。

補　足

ansible.builtin.git モジュールを使うためには、ターゲットノートに git コマンドが必要です。「dnf install git」などでインストールしてください。

関　連

▷ ansible.builtin.git – Deploy software (or files) from git checkouts – Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/git_module.html

▷ Git

https://git-scm.com/

3-3-8　replace で正規表現にマッチする文字列をすべて置換する

キーワード

▷ ansible.builtin.replace モジュール

方　法

ansible.builtin.replace モジュールは、正規表現にマッチするファイル内の複数箇所の文字列をすべて置換します。List 3-53 のプレイブックは、Apache の設定ファイル内の AllowOverride ディレクティブの設定を AllowConfig に変更する例です。

List 3-53　ansible.builtin.replace モジュールの利用例: ./Linux/file_replace_httpd_configure.yml

```
---
- hosts: private_www
  gather_facts: false
  become: true

  tasks:
    - name: enable authconfig
      ansible.builtin.replace:
        path: /etc/httpd/conf/httpd.conf
        regexp: ^(\s*AllowOverride).*
        replace: \1 AuthConfig
        after: <Directory "/var/www/html">
        before: </Directory>
        validate: httpd -t -f %s
```

解　説

ansible.builtin.replace モジュールの主なパラメータは Table 3-21 のとおりです。

Table 3-21　ansible.builtin.replace モジュールの主なパラメータ

パラメータ名	説明
path	処理対象のファイル
regexp	書き換え対象を検索するための正規表現
replace	regexp にマッチした箇所へ書き込む文字列
after	指定した箇所以降のみを処理対象にする
before	指定した箇所以前のみを処理対象にする
validate	変更したファイルの構文テストを行うコマンドを指定

ansible.builtin.replace モジュールは正規表現を使用して文字列置換をするモジュールのため、ansible.builtin.lineinfile モジュールとは異なり、マッチする文字列がなかった場合に新規の文字列は追加しません。

また、ansible.builtin.replace モジュールは regexp パラメータにマッチした部分のみを replace パラメータで指定した文字列に変更します。そのため、検索する正規表現が置換後の文字列の一部にもマッチするような書き方をする場合、再実行には注意が必要です。たとえば、ポート番号設定を「80」から「8080」へ変更するために以下のようなタスクを作成してしまうと、1 回目の実行は期待どおりに「80」だった設定が「8080」に変更されます。しかし、続けて 2 回目の実行すると「port: 80」まではマッチしてしまうため、置換後の文字列は「808080」となってしまいます。このように、使い方によっては冪等性がなくなるので注意してください。

List 3-54　繰り返し実行時に期待どおり動作しない記述

```
1: regexp: 'port: 80'
2: replace: 'port: 8080'
```

この場合は、行頭から行末までマッチするような正規表現を記述してください。

List 3-55　繰り返し実行しても期待どおり動作する記述

```
1: regexp: '^port: 80$'
2: replace: 'port: 8080'
```

after と before パラメータを使用することで、指定した行以降、あるいは指定した行までを処理対象とする範囲を決められるため、ファイル内の複数マッチする対象の中から、文字列置換したい箇所を柔軟に指定できます。

ファイル編集を行うモジュールは validate というパラメータがあります。ファイル構文の正常性を確認するコマンドを validate パラメータに指定することで、もしファイル構文の正常性確認でエラーが発生した場合はファイルを書き換えずに停止する機能があります。正常性を確認するファイルのパスは「`%s`」で指定する必要があります。構文テスト用のコマンドが存在する場合は安全のために指定することを推奨します。たとえば、sshd の設定ファイル（`/etc/ssh/sshd_config`）であれば、「`sshd -t -f %s`」で構文テストができます。使用時は各アプリケーションのマニュアルを確認してください。

関　連

▷ ansible.builtin.replace – Replace all instances of a particular string in a file using a back-referenced regular expression – Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/ansible/builtin/replace_module.html`

3-3-9　lineinfile で文字列を追記または更新する

キーワード

▷ ansible.builtin.lineinfile モジュール

方　法

ansible.builtin.lineinfile モジュールは、regexp パラメータで指定した正規表現にマッチする行を line パ

ラメータで指定した値に書き換えます。List 3-56 のプレイブックは、sudo 実行時のパスワード入力を省略する設定の例です。

List 3-56　ansible.builtin.lineinfile モジュールの利用例: ./Linux/file_lineinfile_sudo_configure.yml

```
1: ---
2: - hosts: linux
3:   gather_facts: false
4:   become: true
5:
6:   tasks:
7:     - name: lineinfile sample
8:       ansible.builtin.lineinfile:
9:         path: /etc/sudoers
10:        regexp: '^%wheel\s+ALL='
11:        line: '%wheel ALL=(ALL) NOPASSWD: ALL'
12:        validate: /usr/sbin/visudo -c -f %s
```

解　説

RHEL 系の Linux では、wheel グループに所属するユーザはデフォルトでは自分のパスワードを入力することで sudo を使ったコマンドの特権実行が可能です。sudoers の設定に NOPASSWD を追加することで、このパスワード入力を省略できます。Debian や Ubuntu では、デフォルトは wheel ではなく sudo グループが使用されます。設定内容の詳細は sudoers(5) のマニュアルを参照してください。

なお、このタスクは特権が必要なファイル編集を行います。初期状態では権限昇格のためのパスワード入力が必要です。そのため、ansible-playbook コマンド実行時に--ask-become-pass（-K）オプションを付与して、昇格に必要なパスワード入力プロンプトを表示させてパスワードを入力するか、ansible_become_password 変数でパスワードを指定してタスクを実行してください（Table 3-22）。

Table 3-22　ansible.builtin.lineinfile モジュールの主なパラメータ

パラメータ名	説明
path	処理対象のファイル
regexp	書き換え対象を検索するための正規表現‡
line	regexp にマッチした行へ書き込む文字列
insertafter	正規表現にマッチした行がある場合、line の文字列を次の行に追加
insertbefore	正規表現にマッチした行がある場合、line の文字列を前の行に追加
backrefs	true にすると後方参照を使用可能
validate	変更したファイルの構文テストを行うコマンドを指定

‡　マッチする行がない場合は insertafter または insertbefore パラメータが基準になります

ansible.builtin.replace モジュールが正規表現にマッチした部分を置換するのに対して、ansible.builtin.lineinfile モジュールは regexp パラメータで指定した正規表現にマッチする行全体を line パラメータに指定した文字列に書き換えます。マッチする行が見つからなかった場合は、line パラメータに指定した行を新規に追記します。追記する場所はデフォルトではファイル末尾です。

新規追加する場所を指定する場合は、insertafter パラメータまたは insertbefore パラメータで指定します。insertafter パラメータは指定した箇所の次の行へ、insertbefore パラメータは指定した箇所の前の行へ追記します。どちらのパラメータも正規表現を指定できますが、ファイル末尾・ファイル先頭を表す場合は EOF（ファイル末尾）、BOF（ファイル先頭）という文字列を指定します。デフォルトは「`insertafter: EOF`」が指定されているため、ファイル末尾へ追記します。ファイル先頭に追記したい場合は、「`insertbefore: BOF`」を指定します。指定に一致する行が複数あった場合は、最後にマッチした行の前後に追記されます。insertafter と insertbefore パラメータはどちらか片方のみ指定できます。

backrefs パラメータに true を指定することで、正規表現の後方参照ができるようになります。デフォルトは無効です。ただし、後方参照の使い方によっては ansible.builtin.replace モジュールの例と同様に、繰り返し実行することで冪等ではなくなる場合があるので注意が必要です。なお、backrefs は、insertafter または insertbefore パラメータと併用できません。

関　連

▷ ansible.builtin.lineinfile – Manage lines in text files– Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/ansible/builtin/lineinfile_module.html`

3-3-10　blockinfile で複数行の文字列を追記または更新する

キーワード

▷ ansible.builtin.blockinfile モジュール

方　法

ansible.builtin.blockinfile モジュールは、複数行のテキストをファイルに追記または更新します。List 3-57 のプレイブックは、BIND の設定ファイルにゾーンファイルのインクルードを行うための複数行の設定を追加する例です。

List 3-57　ansible.builtin.blockinfile モジュールの利用例: ./Linux/file_blockinfile_bind_configure.yml

```
---
- hosts: dnsserver
  gather_facts: false
  become: true

  tasks:
    - name: append zonefile include
      ansible.builtin.blockinfile:
        path: /etc/named.conf
        marker: // example.org zone {mark}
        block: |
          zone "example.org" {
              type master;
              file "/etc/named/named.example.zones";
          };
        validate: named-checkconf %s
```

解　説

BIND のゾーンファイルを新しく作成した際に、BIND の設定ファイルである named.conf にゾーンファイルをインクルードする設定を追加します。設定は複数行になっているため、ansible.builtin.blockinfile モジュールを使用します。

ansible.builtin.replace モジュールや ansible.builtin.lineinfile モジュールでは行単位の処理ですが、複数行のまとまったテキストの追記や更新には ansible.builtin.blockinfile モジュールを使います。

主なパラメータは Table 3-23 のとおりです。

Table 3-23　ansible.builtin.blockinfile モジュールの主なパラメータ

パラメータ名	説明
marker	開始行・終了行のマーカー行
marker_begin	開始行で marker 内の {mark} を置き換える
marker_end	終了行で marker 内の {mark} を置き換える
insertafter	新規追加する場合にマッチした行の次の行に追加
insertbefore	新規追加する場合にマッチした行の前の行に追加
validate	変更したファイルの構文テストを行うコマンドを指定

ansible.builtin.blockinfile モジュールは、追加や更新したい複数行のテキストを「**マーカー行**」を目印に追加する場所を検索し、編集します。マーカー行は開始行と終了行があり、開始行から終了行の間の行を block パラメータで指定したテキストに更新します。マーカー行が見つからない場合は、デフォルトではファイル末尾に追加されますが、開始行と終了行も含めて追加されます。

マーカー行はデフォルトでは「#」で始まる文字列です。この例のように、必要に応じて操作対象のファイルのコメント書式に合わせて、マーカー行を変更します。

marker_begin パラメータと marker_end パラメータは、デフォルトではそれぞれ「BEGIN」と「END」が設定されています。marker パラメータで指定する文字列内の「{mark}」の部分が、開始行と終了行でそれぞれ marker_begin と marker_end に指定した文字列に置き換わります。

marker パラメータを対象ファイルのコメント行として使用するのがセオリーですが、Apache HTTP Server の設定ファイルのように、開始タグと終了タグで囲まれているようなテキストはタグをマーカーとしても使用できます。ただし、同じマーカーがファイル内に複数ある場合は、最後のマーカーが使用されるので注意してください。List 3-58 のような<Directory>の開始と終了を指定した場合、<Directory "/var/www">から直近の</Directory>でなく、ファイル内の最後の</Directory>までの間のテキストを編集します。もし<Directory "/var/www">が複数あった場合は、ファイル内の最後の<Directory "/var/www">が開始位置になります。

List 3-58　設定ファイルの既存内容をマーカーに設定する場合

```
1: marker_begin: '<Directory "/var/www">'
2: marker_end: '</Directory>'
```

insertafter パラメータと insertbefore パラメータについては、ansible.builtin.lineinfile モジュールと同じです。マーカー行が見つからない場合に新規追加する場所を指定します。

関　連

▷ ansible.builtin.blockinfile – Insert/update/remove a text block surrounded by marker lines – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/ansible/builtin/blockinfile_module.html

3-4　Web サーバ構築例

Web サーバの構築において、Basic 認証や証明書作成を行うプレイブックを紹介します。本節では Apache HTTP Server（以下 Apache）を例に認証や証明書を利用する設定を行いますが、それらの設定は Apache 以外のさまざまなプロダクトに応用できます。その他のプロダクトで認証や証明書のファイルを利用する場合は、設定ファイル名などを適宜読み替えてください。

3-4-1　Basic 認証のパスワードファイルを作成する

キーワード

▷ htpasswd コマンド

▷ community.general.htpasswd モジュール

方　法

Basic 認証のためのパスワードファイルを作成するには、community.general.htpasswd モジュールを使用します。

List 3-59　community.general.htpasswd モジュールの利用例: ./Linux/web_basic_auth.yml

```
1: ---
2: - hosts: private_www
3:   gather_facts: false
4:   become: true
5:
6:   tasks:
7:     - name: create htpasswd
8:       community.general.htpasswd:
9:         path: /var/www/html/auth/.htpasswd
10:        name: www-user
11:        password: 'password'
12:        mode: "0644"
```

解　説

community.general.htpasswd モジュールで使用する主なパラメータは Table 3-24 のとおりです。

Table 3-24　htpasswd モジュールで使用する主なパラメータ

パラメータ名	説明
path	作成するパスワードファイル
name	追加するユーザ名
password	設定するパスワード (平文)
mode	ファイルのアクセス権

path パラメータで指定したファイルに、name と password で指定した内容のアカウント情報が作成されます。

ファイルが存在しない場合は新規作成されます。ファイルはすでに存在して指定ユーザの情報がファ

イル内にない場合は追記され、指定ユーザの情報がすでにある場合は内容が更新されます。

応　用

このモジュールは Basic 認証用のパスワードファイルを作成するだけなので、認証を有効にする Web サーバの設定は別途必要です。Apache であれば、認証を有効にするにはサーバワイドの「`httpd.conf`」への設定や、コンテンツディレクトリ単位の「`.htaccess`」への設定を行います[*3]。`.htaccess` ファイルや Web サーバ自体の設定の詳細は、Apache のドキュメントを参照してください。

また、state パラメータに absent を指定すると、指定したユーザのパスワード情報が削除されます。

補　足

このモジュールの使用には、マネージドノードで passlib パッケージが必要です。`pip` コマンドなどでインストールします。

関　連

▷ community.general.htpasswd – manage user files for basic authentication – Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/community/general/htpasswd_module.html`
▷ htpasswd – Manage user files for basic authentication – Apache HTTP Server Version 2.4
`https://httpd.apache.org/docs/current/programs/htpasswd.html`
▷ Authentication and Authorization – Apache HTTP Server Version 2.4
`https://httpd.apache.org/docs/current/howto/auth.html`

3-4-2　HTTPS サーバ用の自己署名証明書を作成する

キーワード

▷ `openssl` コマンド
▷ community.crypto.openssl_privatekey モジュール
▷ community.crypto.openssl_csr モジュール
▷ community.crypto.x509_certificate モジュール

＊3　`.htaccess` ファイルを使用する場合は準備が必要です。
`https://httpd.apache.org/docs/current/howto/auth.html#theprerequisites`

方 法

検証用途やクローズドな環境などで自己署名の証明書を使った HTTPS 通信が必要な場合に、Ansible では以下のモジュールを使用して証明書を作成できます。これらのモジュールを使ったプレイブックの例として「ansible.example.org」という FQDN 用の自己署名証明書を作成します。

- community.crypto.openssl_privatekey
- community.crypto.openssl_csr
- community.crypto.x509_certificate

List 3-60　自己署名のサーバ証明書を作成する例: ./Linux/web_https_selfsigned.yml

```
---
- hosts: private_www
  gather_facts: false
  become: true

  vars:
    cert_dir: /opt/self-cert
    server_fqdn: ansible.example.org

  tasks:
    - name: create directory for cert keys
      ansible.builtin.file:
        path: "{{ cert_dir }}"
        mode: "0755"
        state: directory
        owner: root
        group: root

    - name: create server key
      community.crypto.openssl_privatekey:
        path: "{{ cert_dir }}/server.key"

    - name: create csr
      community.crypto.openssl_csr:
        privatekey_path: "{{ cert_dir }}/server.key"
        path: "{{ cert_dir }}/server.csr"
        common_name: "{{ server_fqdn }}"

    - name: generate certificate
      community.crypto.x509_certificate:
```

```
31:         path: "{{ cert_dir }}/server.crt"
32:         csr_path: "{{ cert_dir }}/server.csr"
33:         privatekey_path: "{{ cert_dir }}/server.key"
34:         provider: selfsigned
```

解　説

証明書作成は、以下の手順で行います。

(1) 秘密鍵作成

(2) CSR（Certificate Signing Request）作成

(3) 自己署名したサーバ証明書作成

◇ 秘密鍵作成

秘密鍵の作成は、community.crypto.openssl_privatekey モジュールを使用します。

主なパラメータは Table 3-25 のとおりです。

Table 3-25　community.crypto.openssl_privatekey モジュールで使用する主なパラメータ

パラメータ名	説明
path	鍵ファイル作成先パス
size	鍵サイズ (省略時は 4096)
force	true にするとファイル存在時に再作成

force パラメータは省略するとデフォルトは false です。その場合、path パラメータで指定したパスに鍵ファイルが存在する場合は再作成しません。

◇ CSR 作成

CSR の作成は community.crypto.openssl_csr モジュールを使用します。

主なパラメータは Table 3-26 のとおりです。

Table 3-26　community.crypto.openssl_csr モジュールで使用する主なパラメータ

パラメータ名	説明
path	CSR ファイル作成先パス
privatekey_path	秘密鍵ファイルのパス
common_name	コモンネーム

作成済みの秘密鍵を privatekey_path パラメータに指定します。

CSR 作成時に指定できる各種パラメータは、country_name、state_or_province_name、locality_name、organization_name など、CLI の `openssl` コマンド使用時と同じものがあります。それぞれの詳細については community.crypto.openssl_csr モジュールのドキュメントに記載されているパラメータの項を参照してください。

common_name パラメータには、クライアントからアクセスする際に使用する FQDN を指定します。このパラメータは `openssl` コマンド使用時の Common Name と同じものです。本項の例では「`ansible.example.org`」としています。

また、秘密鍵にパスフレーズが設定されている場合は、privatekey_passphrase パラメータで指定します。

◇ 証明書作成

証明書の作成は community.crypto.x509_certificate モジュールを使用します。

主なパラメータは Table 3-27 のとおりです。

Table 3-27　community.crypto.x509_certificate モジュールで使用する主なパラメータ

パラメータ名	説明
path	証明書ファイル作成先パス
csr_path	CSR ファイルのパス
privatekey_path	秘密鍵ファイルのパス
provider	署名の方式

自己署名の証明書を作成する場合は、provider パラメータに selfsigned を指定します。

また、秘密鍵にパスフレーズが設定されている場合は、community.crypto.openssl_csr モジュールと同様に privatekey_passphrase パラメータで指定します。

作成した証明書ファイルと秘密鍵は Apache であれば、SSLCertificateFile に証明書ファイル、SSLCertificateKeyFile に秘密鍵ファイルを設定し、証明書ファイルをクライアントへ配布すれば使用できます。詳細はドキュメントを参照ください。

応　用

認証局を自己署名で作成し、この認証局で署名したサーバ証明書の作成もできます。自己署名の認証局は、本項の自己署名証明書の秘密鍵、CSR、証明書の作成と同じ手順で community.crypto.openssl_privatekey モジュール、community.crypto.openssl_csr モジュール、community.crypto.x509_certificate モジュールを使用して作成します。

作成した認証局を使ってサーバ証明書に署名するには、サーバ証明書の作成で community.crypto.x509_certificate モジュールを使用するときの provider パラメータに owncа を指定します。加えて owncа_path パラメータと owncа_privatekey_path パラメータで、認証局の証明書ファイルと秘密鍵ファイルのパスを指定します。

List 3-61　自己署名の認証局で署名するタスク

```
1: - name: generate certificate with own CA
2:   community.crypto.x509_certificate:
3:     path: "{{ cert_dir }}/server.crt"
4:     csr_path: "{{ cert_dir }}/server.csr"
5:     owncа_path: "{{ cert_dir }}/ca.crt"
6:     owncа_privatekey_path: "{{ cert_dir }}/ca.key"
7:     provider: owncа
```

補　足

利用には cryptography パッケージか pyOpenSSL が必要です。pip などでインストールします。

関　連

▷ community.crypto.openssl_privatekey – Generate OpenSSL private keys – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/community/crypto/openssl_privatekey_module.html

▷ community.crypto.openssl_csr – Generate OpenSSL Certificate Signing Request (CSR) – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/community/crypto/openssl_csr_module.html

▷ community.crypto.x509_certificate – Generate and/or check OpenSSL certificates – Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/community/crypto/x509_certificate_module.html

▷ mod_ssl – Apache HTTP Server Version 2.4
https://httpd.apache.org/docs/current/mod/mod_ssl.html

3-4-3　Let's Encrypt を使った証明書で HTTPS サーバを構築する

キーワード

▷ ACME

▷ Let's Encrypt

▷ community.crypto.acme_certificate モジュール

方　法

無料で SSL/TLS 証明書を作成できる Let's Encrypt のサービスを使用して証明書を作成するには、community.crypto.acme_certificate モジュールを使用します。

Apache を使って、ACME の HTTP-01 チャレンジタイプで証明書を作成するプレイブックは List 3-62 のとおりです。このプレイブックは HTTP-01 チャレンジで使用する Web サーバのインストールも含んでいます。

List 3-62　community.crypto.acme_certificate モジュールの利用例: ./Linux/web_https_acme.yml

```
---
- hosts: public_www
  gather_facts: false
  become: true

  vars:
    cert_dir: /opt/acme_cert
    server_fqdn: ansible.example.org

  tasks:
    - name: install httpd
      ansible.builtin.dnf:
        name:
          - httpd
          - mod_ssl
        state: present

    - name: start httpd
      ansible.builtin.service:
        name: httpd
        state: started

    - name: create directory
      ansible.builtin.file:
```

```
25:         state: directory
26:         path: "{{ cert_dir }}/{{ server_fqdn }}"
27:
28:     - name: (1) create account private key
29:       community.crypto.openssl_privatekey:
30:         path: "{{ cert_dir }}/{{ server_fqdn }}/account.key"
31:
32:     - name: (2) create server key
33:       community.crypto.openssl_privatekey:
34:         path: "{{ cert_dir }}/{{ server_fqdn }}/{{ server_fqdn }}.key"
35:
36:     - name: (3) create csr
37:       community.crypto.openssl_csr:
38:         path: "{{ cert_dir }}/{{ server_fqdn }}/{{ server_fqdn }}.csr"
39:         privatekey_path: "{{ cert_dir }}/{{ server_fqdn }}/{{ server_fqdn }}.key"
40:         common_name: "{{ server_fqdn }}"
41:
42:     - name: (4) create challenge for {{ server_fqdn }} using a account key file.
43:       community.crypto.acme_certificate:
44:         account_key_src: "{{ cert_dir }}/{{ server_fqdn }}/account.key"
45:         account_email: "{{ email }}"
46:         csr: "{{ cert_dir }}/{{ server_fqdn }}/{{ server_fqdn }}.csr"
47:         dest: "{{ cert_dir }}/{{ server_fqdn }}/{{ server_fqdn }}.crt"
48:         acme_directory: https://acme-v02.api.letsencrypt.org/directory
49:         acme_version: 2
50:         terms_agreed: true
51:       register: acme_challenge
52:
53:     - name: create directory for token file
54:       ansible.builtin.file:
55:         state: directory
56:         path: "/var/www/html/{{ acme_challenge['challenge_data'][server_fqdn]⇒
57: ['http-01']['resource'] | ansible.builtin.dirname }}"
58:       when: acme_challenge is changed and server_fqdn in acme_challenge⇒
59: ['challenge_data']
60:
61:     - name: (5) create token file to server
62:       ansible.builtin.copy:
63:         dest: "/var/www/html/{{ acme_challenge['challenge_data'][server_fqdn]⇒
64: ['http-01']['resource'] }}"
65:         content: "{{ acme_challenge['challenge_data'][server_fqdn]['http-01']⇒
66: ['resource_value'] }}"
67:       when: acme_challenge is changed and server_fqdn in acme_challenge⇒
68: ['challenge_data']
69:
```

```
70:     - name: (6) validated challenge and retrieve certificate
71:       community.crypto.acme_certificate:
72:         account_key_src: "{{ cert_dir }}/{{ server_fqdn }}/account.key"
73:         account_email: "{{ email }}"
74:         csr: "{{ cert_dir }}/{{ server_fqdn }}/{{ server_fqdn }}.csr"
75:         dest: "{{ cert_dir }}/{{ server_fqdn }}/{{ server_fqdn }}.crt"
76:         acme_directory: https://acme-v02.api.letsencrypt.org/directory
77:         acme_version: 2
78:         terms_agreed: true
79:         data: "{{ acme_challenge }}"
```

解　説

Let's Encrypt で証明書を作成するには、ACME というプロトコル[*4]を使用します。Ansible では、community.crypto.acme_certificate モジュールを使うことで、この ACME プロトコルを使った証明書を作成できます。

本項では HTTP-01 チャレンジの場合について説明します。この方式の仕様として、証明書作成中に構築するサーバからのチャレンジ要求のレスポンスとして Let's Encrypt から 80/TCP で HTTP アクセスできる必要があります。作成する証明書の FQDN で外部からの疎通確認をするため、DNS の設定も事前に必要です。詳細は Let's Encrypt のドキュメント[*5]を参照してください。

大まかな流れは以下のとおりです。番号は冒頭のプレイブックのタスク名の番号になっており、(4) と (6) の手順で計 2 回、本モジュールを使用するタスクが必要です。

(1) Let's Encrypt 用のアカウントキーファイルを作成
(2) サーバ証明書用の秘密鍵を作成
(3) サーバ証明書の CSR を作成
(4) アカウントキーとサーバ証明書の CSR をパラメータに指定して Let's Encrypt に ACME のチャレンジ要求
(5) (4) のチャレンジ要求の戻り値を使ってトークンファイルを作成
(6) 最後にチャレンジを検証し、証明書を取得

(1) のアカウントキーファイルの作成と、(2) および (3) のサーバ証明書の秘密鍵および CSR の作成につ

＊4　Automatic Certificate Management Environment
　　https://datatracker.ietf.org/doc/html/rfc8555
＊5　チャレンジのタイプ - Let's Encrypt - フリーな SSL/TLS 証明書
　　https://letsencrypt.org/ja/docs/challenge-types/

いては「3-4-2 HTTPS サーバ用の自己署名証明書を作成する」と同様に community.crypto.openssl_privatekey モジュールと community.crypto.openssl_csr モジュールを使用します。

community.crypto.acme_certificate モジュールは、「チャレンジの要求のタスク (4)」と「チャレンジの結果から証明書情報を取得するタスク (6)」の、2 回の実行が必要です。

HTTP-01 チャレンジの場合に使用する主なパラメータは Table 3-28 のとおりです。

Table 3-28　HTTP-01 チャレンジで使用する community.crypto.acme_certificate モジュールの主なパラメータ

パラメータ名	説明
account_key_src	openssl_privatekey モジュールで作成するアカウントキーファイル
account_email	アカウントに紐付けるメールアドレス
csr	サーバ証明書の CSR
dest	作成される署名済みサーバ証明書
acme_directory	ACME Directory(URL）を指定
acme_version	ACME エンドポイントのバージョン
terms_agreed	利用規約の同意フラグ
data	チャレンジに使用するデータ

以下では、それぞれのパラメータについて説明します。

- account_key_src パラメータ

  前述のアカウントキーファイルのパスを指定します。account_email パラメータには証明書の有効期限の通知用メールアドレスを指定します。

- csr パラメータ

  前述のサーバ証明書の CSR ファイルのパスを指定します。

- dest パラメータ

  作成する証明書ファイルの出力先パスを指定します。本モジュールを使用するタスクの 2 回目の「チャレンジ結果から証明書情報を取得するタスク (6)」の実行の結果、このパスへ Let's Encrypt で発行された証明書ファイルが作成されます。

- acme_directory パラメータ

  ACME を使った証明書発行のサービスの API エンドポイントとなる URL を指定します。Let's Encrypt の ACME v2 の場合は以下の URL を指定します。その他のサービスを利用する場合は各サービスのドキュメントなどを確認してください。

```
https://acme-v02.api.letsencrypt.org/directory
```

- acme_version パラメータ

  acme_directory パラメータで指定したエンドポイントの ACME バージョンを指定します。デフォルト値は 1 ですが、Let's Encrypt では v1 はサービス終了しているため 2 を指定します。

- terms_agree パラメータ

  このパラメータは利用許諾です。acme_version を 2 に指定する場合は true に指定する必要があります。

- data パラメータ

  本モジュールを使用する 1 回目のタスクのチャレンジ要求の結果を、2 回目のタスク実行の入力に指定します。1 回目のタスク実行時の結果を register ディレクティブを使用して保持し、2 回目のタスクで参照します。

  1 回目のチャレンジ要求のタスク実行では、タスク実行の戻り値にトークンなどが含まれます。この内容をもとに「`http://<FQDN>/.well-known/acme-challenge/<TOKEN>`」という URL で外部からアクセスできるように、戻り値のリソース情報のファイル作成します。Let's Encrypt はこの URL にアクセスしてファイルの内容を検証し、証明書を作成します。

応　用

作成した一連のタスクは、再度実行することで証明書の更新ができます。Let's Encrypt で発行される証明書は 90 日の有効期間が設定されます。期限が切れる前に再実行し、90 日間の有効期間が設定された新しい証明書を作成することで、証明書の有効期間を更新します。

ただし、無条件で再実行のたびに新しい証明書が作成されるわけではありません。community.crypto.acme_certificate モジュールを使った証明書作成は、既存の証明書の期限の残り日数によって動作するかが決定します。モジュール実行時の戻り値内の、cert_days に既存の証明書の有効期限の残り日数がセットされます。この cert_days の日数が remaining_days パラメータの値より少ない場合に、新しい証明書を作成できるようになっています。remaining_days パラメータのデフォルト値は 10 のため、毎日定期的に実行した場合は、証明書の期限が残り 10 日未満になったタイミングで新しい証明書が作成されます。remaining_days の値を運用に合わせて指定することで、任意のサイクルで証明書の更新ができます。

また、残り日数に関係なく証明書を更新するには、force パラメータを true にします。この指定をすることで強制的に証明書を作成します。

Apache であれば Web サーバ設定の証明書ファイルのパスに取得した証明書ファイルを指定し、Apache プロセスの再起動をすれば使用できます。

List 3-63　作成した証明書を Apache に設定するタスク例

```
- name: replace ssl.conf configure
  ansible.builtin.replace:
    path: /etc/httpd/conf.d/ssl.conf
    regexp: "^{{ item.key }}.*"
    replace: "{{ item.key }} {{ item.file }}"
  loop:
    - key: SSLCertificateFile
      file: "{{ cert_dir }}/{{ server_fqdn }}/{{ server_fqdn }}.crt"
    - key: SSLCertificateKeyFile
      file: "{{ cert_dir }}/{{ server_fqdn }}/{{ server_fqdn }}.key"

- name: restart httpd
  ansible.builtin.service:
    name: httpd
    state: restarted
```

補　足

community.crypto.acme_certificate モジュールの実行には cryptography パッケージが必要なので pip でインストールします。

ACME プロトコルの HTTP-01 チャレンジを使った証明書作成には、FQDN で名前解決できることとインターネットからの 80/TCP アクセスが必要なので、アクセス制限と DNS 設定を事前に済ませておく必要があります。なお、本項では外部からのアクセスを許可した AWS の EC2 で動作検証を行いました。

関　連

▷ community.crypto.acme_certificate – Create SSL/TLS certificates with the ACME protocol – Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/community/crypto/acme_certificate_module.html

▷ rfc8555 Automatic Certificate Management Environment (ACME)

https://datatracker.ietf.org/doc/html/rfc8555

▷ チャレンジのタイプ - Let's Encrypt - フリーな SSL/TLS 証明書

https://letsencrypt.org/ja/docs/challenge-types/

▷ End of Life Plan for ACMEv1 – API Announcements - Let's Encrypt Community Support

https://community.letsencrypt.org/t/end-of-life-plan-for-acmev1/88430

3-5　コマンドの実行

本節では、OS コマンドなどを直接実行して自動化するためのプレイブックを紹介します。モジュールの使い方に加えて、実行ステータスが実態に即した出力となるためのポイントや、チェックモード時の注意点について説明します。

3-5-1　Linux コマンドを実行する

キーワード

▷ ansible.builtin.command モジュール

▷ ansible.builtin.shell モジュール

方　法

Linux コマンドを実行するには、ansible.builtin.command モジュールか、ansible.builtin.shell モジュールを使用します。

List 3-64　ansible.builtin.command モジュールの利用例: ./Linux/command_exec.yml

```
1: ---
2: - hosts: linux
3:   gather_facts: false
4:
5:   tasks:
6:     - name: execute command
7:       ansible.builtin.command:
8:         cmd: nmcli connection show
```

解　説

ansible.builtin.command モジュールで使用する主なパラメータは Table 3-29 のとおりです。

Table 3-29　ansible.builtin.command モジュールの主なパラメータ

パラメータ名	説明
cmd	実行するコマンドライン文字列
argv	cmd の替わりに指定できるコマンドとオプションのリスト
chdir	実行ディレクトリ
stdin	コマンドに対する標準入力
creates	ここで指定するファイルが存在する場合はコマンド実行しない

removes	ここで指定するファイルが存在しない場合はコマンド実行しない

List 3-65 のように YAML のネストしないマッピングの「キー: 値」の書式だけを使って、モジュール指定行の値部分にコマンド文字列を書くこともできます。コマンド実行にディレクトリなどのパラメータ指定が不要な場合は、簡潔に記述できます。

List 3-65　モジュール名とコマンドを「キー: 値」の書式だけで指定した記述: ./Linux/command_exec.yml

```
1:  tasks:
2:    - name: execute command
3:      ansible.builtin.command: nmcli connection show "System ens192"
```

argv は、cmd の代わりに使用するコマンド指定パラメータです。cmd がスペースで区切ってコマンドラインで実行する場合と同様の文字列を指定するのに対して、argv はコマンドと引数をリストで指定します。リスト指定することで意図しないスペースによって引数が分断されるのを防ぐことができます。

List 3-66　argv パラメータにリスト形式で実行コマンドを指定する記述: ./Linux/command_exec.yml

```
1:    - name: execute command
2:      ansible.builtin.command:
3:        argv:
4:          - nmcli
5:          - connection
6:          - show
7:          - System ens192
```

creates パラメータと removes パラメータは指定したファイルが存在するかどうかをチェックします。両方を指定した場合は、どちらかの条件に合致すればコマンドは実行されません。コマンドが実行されない場合は Ansible の実行ステータスは ok となります。コマンド実行の結果ファイルが作成、あるいは削除される場合に本パラメータを指定することで、不要なタスク実行を抑制できます。

ansible.builtin.command モジュールの実行結果で主に使用される戻り値は Table 3-30 のとおりです。

stdout と stdout_lines は内容は同じです。コマンドの出力結果に対し、行ごとに値をチェックしたい場合などは、stdout_lines を使うと便利です。

Table 3-30　ansible.builtin.command モジュールの主な戻り値

値	説明
rc	コマンドの戻り値
stdout	標準出力 (複数行文字列)
stdout_lines	標準出力 (1 行 1 要素のリスト)
stderr	標準エラー出力 (複数行文字列)
stderr_lines	標準エラー出力 (1 行 1 要素のリスト)
start	コマンドの実行開始時刻
end	コマンドの実行完了時刻

応　用

ansible.builtin.command モジュールや ansible.builtin.shell モジュールを使用する上での注意点や、タスクのディレクティブと組み合わせて「**より正しい**」タスクを作るためのポイントについて解説します。

◇ 冪等性の考慮

ansible.builtin.command モジュールや ansible.builtin.shell モジュールの特徴は、タスクの実行ステータスが常に changed になります。実行するコマンドが参照処理のみで何も変更を生じない場合で changed になるのがふさわしくない場合は、changed_when ディレクティブを使用し、changed ではなく ok になるように調整するとよいでしょう。

◇ チェックモードの対応

ほかの特徴として、Ansible をチェックモードで動作させた場合[*6]に処理がスキップされ何も実行されません。タスクの実行結果を register ディレクティブの使用で保持し次のタスクで参照するような場合、スキップ時は戻り値の rc や標準出力の stdout などの値は戻り値に含まれないため、参照すると未定義エラーとなります。スキップ時にもタスクの実行結果が必要な使い方をする場合は、check_mode ディレクティブに false を指定し、チェックモードの実行時でも強制的にタスクを実行させる必要があります。

ただし、デフォルトではチェックモードに対応していないこれらのモジュールは、creates パラメータか removes パラメータを指定している場合に限り、ファイルの有無を確認することでチェックモードをサポートします。その場合タスクの戻り値の rc はゼロになり、標準出力などは固定の文字列がセットされます。

＊6　ansible-playbook のコマンドオプションに「--check」を追加して実行。

◇ タスクの実行条件の指定

コマンドの実行要否を前段のタスクの実行結果で判断したい場合は、when ディレクティブを使った条件指定や、notify と handlers を使った後処理としてタスクを作成します。

◇ エラーのハンドリング

エラーについては、実行したコマンドの戻り値がゼロでない場合にタスクの実行は失敗となり Ansible の処理は停止します。実行するコマンドの結果が非ゼロの場合も後処理が必要など、結果をハンドリングしたい場合は、failed_when ディレクティブを使用して失敗の条件を指定します。

補　足

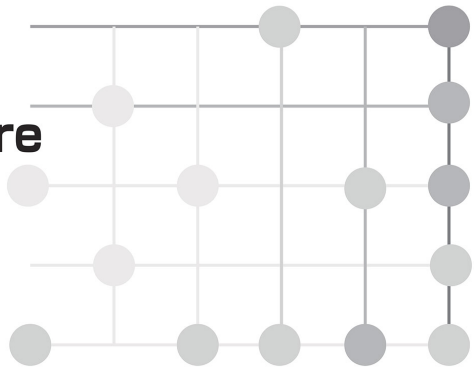
ansible.builtin.shell モジュールは、シェルで許可されていることは実行可能です。そのため、実行するコマンドの引数などに外部からの入力を使用する場合は、Ansible の実行者が入力に記号類を含めることでマネージドノードで想定しないコマンドが実行される可能性などがあり注意が必要です。このような場合は、安全にコマンドを実行できる ansible.builtin.command の使用を推奨します。パイプやリダイレクトなどのシェルの機能が不要な場合は、基本的には ansible.builtin.command を使うようにしてください。ドキュメントの Note の項[*7]も参照してください。

関　連

▷ ansible.builtin.command – Execute commands on targets – Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/ansible/builtin/command_module.html`
▷ ansible.builtin.shell – Execute shell commands on targets – Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/ansible/builtin/shell_module.html`

＊7　`https://docs.ansible.com/ansible/latest/collections/ansible/builtin/shell_module.html#notes`

第４章
VMware



Ansible では VMware に対応したモジュールがコレクションとして提供されており、VMware vSphere 環境の自動化を実現できます。VMware モジュールを使い ESXi ホスト、vCenter Server、ネットワーク、データストアなど vSphere 環境の設定や仮想マシンデプロイなどの自動化をすることで初期構築や運用を効率よく行うことができます。

VMware には community.vmware と vmware.vmware_rest の 2 つのコレクションがありますが、本章では community.vmware を使用した自動化のプレイブックを紹介します。

4-1 概要

各プレイブックの紹介に先立ち、Ansible を vSphere 環境で使用するための前提として、必要な準備および環境について説明します。

4-1-1 ライセンスの必要性

VMware コレクションのモジュールを使用して vSphere 環境を自動化するには、ESXi および vCenter Server の有料ライセンスが必要になります。community.vmware コレクションのモジュールは vSphere Web Services API[*1]を利用しており、ESXi ライセンスの無償版を使用する場合は API が読み取り専用になるため、オブジェクトの作成、削除や更新をする操作は実行できません。

非商用環境の個人検証目的であれば、VMware User Group（VMUG）の Advantage メンバ[*2]に登録してサブスクリプションを購入することで、検証用ライセンスを入手できます。サブスクリプションの価格については VMUG メンバーシップ[*3]のサイトを確認してください。

ESXi および vCenter Server は標準で 60 日間の評価期間があり、その期間中でもモジュールの実行は可能ですが 60 日経過するとモジュールの実行ができなくなるため注意が必要です。

4-1-2 環境の準備

community.vmware コレクションのモジュールを利用するために必要な、コントロールノード側の準備について説明します。

■ コントロールノード側の準備

コントロールノードから ESXi ホストまたは vCenter Server の操作をするためには、Python パッケージ「pyvmomi[*4]」「vSphere Automation SDK for Python[*5]」が必要です。コントロールノード側の「`ansible`」

＊ 1 vSphere Web Services API
`http://code.vmware.com/apis/968/vsphere`
＊ 2 VMware User Group（VMUG）の Advantage メンバ
`http://www.vmug.com/membership/evalexperience/`
＊ 3 VMUG メンバーシップ
`http://my.vmug.com/joinapi__membershiplist?id=a204x000000toAZAAY&order=2`
＊ 4 Python パッケージ「pyvmomi
`http://github.com/vmware/pyvmomi`
＊ 5 vSphere Automation SDK for Python
`http://github.com/vmware/vsphere-automation-sdk-python`

をインストールした Python 環境に追加でインストールします。

List 4-1　community.vmware コレクションのモジュールを実行するために必要な Python パッケージのインストール実行例

```
$ pip install pyvmomi git+https://github.com/vmware/vsphere-automation-sdk-python.git
```

4-1-3　community.vmware コレクションのモジュールの基本仕様

community.vmware コレクションに含まれるモジュールの基本的な仕様について説明します。

■ モジュール一覧と分類

community.vmware コレクションのモジュール名からどういう目的で使用するモジュールなのかをある程度知ることができます。

モジュールの主な命名規則は Table 4-1 のとおりです。

Table 4-1　community.vmware コレクションのモジュール名の主な命名規則

モジュール名の開始	説明
vca_	vCloud Director のモジュール。VMware のプロジェクトで vCloud Director のモジュール‡ が開発されているため本モジュールは将来的に削除される
vcenter_	vCenter Server 関連のモジュール。vmware_vcenter_というモジュールも存在する
vmware_	vmware_の次に続く単語で目的が分類されている。たとえば vmware_host だと ESXi の操作または情報収集をするモジュール
vsphere_	vCenter Server が認識しているデータストアに対してファイルのアップロードおよびフォルダやファイルの作成、削除をするモジュール

‡　vCloud Director のモジュール　http://github.com/vmware/ansible-module-vcloud-director

4-1-4　本章の前提構成

4-2 以降で掲載する各プレイブックの前提となる環境や、インベントリ、変数定義ファイルについて説明します。

■ 環境

ESXi ホストおよび vCenter Server をマネージドノードとした環境です。コントロールノードからマネージドノードへは HTTPS 接続ができる状態とします。

動作確認に使用した ESXi および vCenter Server のバージョンは Table 4-2 のとおりです。

Table 4-2　動作確認使用した ESXi、vCenter Server バージョン

項目	バージョン	ビルド
ESXi	7.0.0	16324942
vCenter Server Appliance	7.0.0	16386335

利用する community.vmware コレクションのバージョンは 1.11.0 です。

Column　community.vmware と community.vmware_rest の違い

VMware に対応したコレクションは、実行対象の API の違いによって community.vmware と vmware.vmware_rest の 2 つがあります。

- community.vmware コレクション

  community.vmware コレクションはもともと Ansible のワンパッケージで提供されていた既存のモジュールをまとめたコレクションです。管理、開発は Ansible コミュニティの VMware ワーキンググループに参加している有志たちによってされています。このコレクションの大部分は VMware 社が OSS として開発している Python 用 SOAP SDK の「pyvmomi」をベースにしており、実行対象となる API は SOAP（Simple Object Access Protocol）API になります。

- community.vmware_rest コレクション

  community.vmware_rest コレクションは新しく追加されたコレクションです。管理、開発は Red Hat 社および Ansible コミュニティの人たちによってされています。このコレクションは vCenter Server 6.5 から提供されている vSphere REST（Representational State Transfer）API を実行対象としており、SDK を使用しなくても利用可能となりました。

このようにコレクションは SOAP と REST で分かれました。使い分けに関してはとくになく利用者の環境や目的に合った方を選択していただければと思います。最後に、vSphere REST API は vCenter Server 6.5 から進化を続けており、API で操作可能な範囲は SOAP の方が広いため community.vmware コレクションの方が機能が豊富です。しかし、community.vmware_rest コレクションは cloud.common コレクション[*6]と連携して高速化を図るための工夫や、将来的に vSphere REST API 拡張によって適用範囲も広がることが期待できます。

■ インベントリと変数ファイル

マネージドノードを定義するインベントリファイルは List 4-2 のとおりです。ESXi ホストと vCenter Server がそれぞれのグループに所属しています。ESXi ホストおよび vCenter Server には、コントロールノードから HTTPS で接続して vSphere Web Services API を実行するため、コネクションプラグインは local を指定しています。

List 4-2　インベントリ: ./VMware/inventory.ini

```
1: [esxi]
2: esxi-001.local
3:
4: [vcenter]
5: vcenter-coock-book.local
6:
7: [all:vars]
8: ansible_connection=local
9: ansible_python_interpreter=環境に合わせて使用するpythonのパスを指定する
```

認証情報やライセンス情報はそれぞれのグループ変数に定義しています。ファイル名は、インベントリファイルで定義したグループ名に合わせています。

List 4-3　ESXi ホストのグループ変数定義ファイル: ./VMware/group_vars/esxi

```
1: ---
2: esxi_username: root
3: esxi_password: ESXiのパスワードを指定する
```

List 4-4　vCenter Server のグループ変数定義ファイル: ./VMware/group_vars/vcenter

```
1: ---
2: vcenter_username: administrator@vsphere.local
3: vcenter_password: vCenter Serverのパスワードを指定する
4: vcenter_license: vCenter Serverのライセンスキーを指定する
```

＊6　cloud.common コレクション
http://github.com/ansible-collections/cloud.common

```
5: esxi_license: ESXiのライセンスキーを指定する
6: esxi_password: ESXiのパスワードを指定する
```

4-2　ESXi 設定

ESXi ホストを対象としたユーザ管理や各種設定などを行うためのプレイブックを紹介します。

4-2-1　ESXi の情報を取得する

キーワード

▷ community.vmware.vmware_host_facts モジュール

方　法

ESXi の情報を取得するには community.vmware.vmware_host_facts モジュールを利用します。本モジュールでは ESXi のサマリ情報や vSphere Web Services API のプロパティを直接指定して情報を取得できます。

List 4-5 の例では ESXi のサマリ情報を取得して表示します。

List 4-5　community.vmware.vmware_host_facts モジュールの利用例: ./ESXi/esxi_host_facts.yml

```
 1: ---
 2: - hosts: esxi
 3:   gather_facts: false
 4:   tasks:
 5:     - name: gather host facts
 6:       community.vmware.vmware_host_facts:
 7:         hostname: "{{ inventory_hostname }}"
 8:         username: "{{ esxi_username }}"
 9:         password: "{{ esxi_password }}"
10:         validate_certs: false
11:         schema: summary
12:       register: host_facts
13:
14:     - name: display host facts
15:       debug:
16:         msg: "{{ host_facts.ansible_facts }}"
```

解　説

community.vmware.vmware_host_facts モジュールの主なパラメータは Table 4-3 のとおりです。

Table 4-3　community.vmware.vmware_host_facts モジュールの主なパラメータ

パラメータ名	説明
schema	アウトプットする情報のスキーマ（summary、vsphere）を指定する
properties	schema が vsphere の場合、取得する情報のプロパティをリストで指定する

プレイブックの例ではサマリとして ESXi の一覧情報を取得しています。サマリでは、ESXi のバージョンやビルド番号およびハードウェア情報などが取得できます。

List 4-6 はサマリ情報で取得した ESXi バージョンと CPU 情報の例です。

List 4-6　サマリ情報の例

```
ok: [esxi-001.local] => {
    "msg": {
...（ 略 ）...
        "ansible_distribution_build": "16324942",
        "ansible_distribution_version": "7.0.0",
...（ 略 ）...
        "ansible_processor": "Intel(R) Xeon(R) CPU D-1528 @ 1.90GHz",
        "ansible_processor_cores": 2,
...（ 略 ）...
```

応　用

schema パラメータが summary の場合、取得できる情報はテンプレートとして固定化されているためサマリ以上の情報を取得できません。そこで、schema パラメータを vsphere に指定し properties パラメータを使用することで、vSphere Web Services API のプロパティを指定できるようになるため柔軟に情報が取得できます。

List 4-7 の例では ESXi が認識しているホストバスアダプタの一覧を取得して表示しています。

List 4-7　ホストバスアダプタの一覧を取得する例: ./ESXi/esxi_host_facts_schema_vsphere.yml

```
1: ---
2: - hosts: esxi
3:   gather_facts: false
4:   tasks:
5:     - name: gather host facts with vsphere schema
```

```
      community.vmware.vmware_host_facts:
        hostname: "{{ inventory_hostname }}"
        username: "{{ esxi_username }}"
        password: "{{ esxi_password }}"
        validate_certs: false
        schema: vsphere
        properties:
          - config.storageDevice.hostBusAdapter
      register: host_facts

    - name: display host facts
      debug:
        msg: "{{ host_facts.ansible_facts.config.storageDevice.hostBusAdapter }}"
```

List 4-8　ESXi が認識しているホストバスアダプタの例

```
ok: [esxi-001.local] => {
    "msg": [
        {
            "_vimtype": "vim.host.ParallelScsiHba",
            "bus": 3,
            "device": "vmhba0",
            "driver": "pvscsi",
            "key": "key-vim.host.ParallelScsiHba-vmhba0",
            "model": "PVSCSI SCSI Controller",
            "pci": "0000:03:00.0",
            "status": "unknown",
            "storageProtocol": "scsi"
        },
...（略）...
```

関　連

▷ community.vmware.vmware_host_facts モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_host_facts_module.html

4-2-2　データストアをマウントする

キーワード

▷ community.vmware.vmware_host_datastore モジュール

方 法

ESXi に対してデータストアのマウントをするには community.vmware.vmware_host_datastore モジュールを利用します。本モジュールでは VMFS として利用するストレージデバイスまたは NFS のマウント操作ができます。

ここでは、VMFS と NFS のマウント方法について紹介します。List 4-9 の例では ESXi に対してストレージデバイスを VMFS データストアとしてマウントします。

List 4-9　community.vmware.vmware_host_datastore モジュール: ./ESXi/esxi_mount_vmfs_volume.yml

```
---
- hosts: esxi
  gather_facts: false
  tasks:
    - name: mount VMFS volume as a datastore
      community.vmware.vmware_host_datastore:
        hostname: "{{ inventory_hostname }}"
        username: "{{ esxi_username }}"
        password: "{{ esxi_password }}"
        validate_certs: false
        esxi_hostname: "{{ inventory_hostname }}"
        datastore_name: vmfs_ds01
        datastore_type: vmfs
        vmfs_device_name: mpx.vmhba0:C0:T1:L0
        vmfs_version: 6
        state: present
```

List 4-10 の例では ESXi に対して NFS のボリュームをデータストアとしてマウントします。

List 4-10　community.vmware.vmware_host_datastore モジュール: ./ESXi/esxi_mount_nfs_volume.yml

```
---
- hosts: esxi
  gather_facts: false
  tasks:
    - name: mount NFS volume as a datastore
      community.vmware.vmware_host_datastore:
        hostname: "{{ inventory_hostname }}"
        username: "{{ esxi_username }}"
        password: "{{ esxi_password }}"
        validate_certs: false
        esxi_hostname: "{{ inventory_hostname }}"
        datastore_name: nfs_ds01
```

```
13:         datastore_type: nfs
14:         nfs_server: 192.168.10.58
15:         nfs_path: /nfs_volume
16:         nfs_ro: false
17:         state: present
```

解　説

community.vmware.vmware_host_datastore モジュールの主なパラメータは Table 4-4 のとおりです。

Table 4-4　community.vmware.vmware_host_datastore モジュールでデータストアを操作する際の共通パラメータ

パラメータ名	説明
esxi_hostname	ESXi ホスト名を指定する
datastore_name	ESXi に登録またはすでに登録されているデータストア名を指定する
datastore_type	データストアのタイプを指定する（nfs、nfs41、vmfs）
state	データストアのマウント（present）またはアンマウント（absent）を指定する

Table 4-5　community.vmware.vmware_host_datastore モジュールでストレージデバイスをマウントする主なパラメータ

パラメータ名	説明
vmfs_device_name	ESXi が認識しているストレージデバイスの識別子を指定する
vmfs_version	VMFS のバージョンを指定する

Table 4-6　community.vmware.vmware_host_datastore モジュールで NFS をマウントする主なパラメータ

パラメータ名	説明
nfs_server	マウントする NFS サーバを指定する
nfs_path	NFS がマウント許可をしているディレクトリのパスを指定する
nfs_ro	読み込みのみでマウントするかどうかを指定する

VMFS をマウントするときに vmfs_device_name パラメータに指定する識別子は、以下のパラメータから確認できます。

- vCenter Server にログインして確認する場合は、ESXi ホストを選択して「設定」タブを開き「ストレージ デバイス」に表示されているストレージデバイスのプロパティの識別子から確認できます
- ESXi にログインして確認する場合は、「ストレージ」を選択して「デバイス」タブを開くと「デバ

イス名（識別子）」としてストレージデバイスが表示されているので、丸かっこから確認できます

関　連

▷ community.vmware.vmware_host_datastore モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_host_datastore_module.html

4-2-3　ファイアウォールの設定をする

キーワード

▷ community.vmware.vmware_host_firewall_manager モジュール

方　法

ESXi のファイアウォール設定をするには community.vmware.vmware_host_firewall_manager モジュールを利用します。本モジュールではサービスに対してアクセス制御ができます。

List 4-11 の例では SSH の接続を許可する IP アドレスとネットワークの設定をします。

List 4-11　community.vmware.vmware_host_firewall_manager モジュールの利用例: ./ESXi/esxi_firewall.yml

```
---
- hosts: esxi
  gather_facts: false
  tasks:
    - name: set ESXi firewall rule
      community.vmware.vmware_host_firewall_manager:
        hostname: "{{ inventory_hostname }}"
        username: "{{ esxi_username }}"
        password: "{{ esxi_password }}"
        validate_certs: false
        esxi_hostname: "{{ inventory_hostname }}"
        rules:
          - name: sshServer
            enabled: true
            allowed_hosts:
              all_ip: false
              ip_address:
                - 192.168.10.1
              ip_network:
                - 192.168.20.0/24
```

解　説

community.vmware.vmware_host_firewall_manager の主なパラメータは Table 4-7 のとおりです。

Table 4-7　community.vmware.vmware_host_firewall_manager モジュールの主なパラメータ

パラメータ名			説明
esxi_hostname			ESXi ホスト名を指定する
rules			ファイアウォールのルールをリストで指定する
	name		設定するファイアウォールのルール名を指定する
	enabled		ルールの有効化無効化を指定する
	allowed_hosts		許可するホストをリストで指定する
		all_ip	すべてのホストから許可するかどうかを指定する、接続元を制御する場合は false を指定する
		ip_address	許可する IP アドレスをリストで指定する
		ip_network	許可するネットワークの CIDR をリストで指定する

プレイブックの例では接続元の IP アドレスおよびネットワークを制御するため all_ip を false にして明示的に指定しています。もし、すべての接続元から許可する場合は all_ip を true にしてください。

関　連

▷ community.vmware.vmware_host_firewall_manager モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_host_firewall_manager_module.html

4-2-4　サービスの操作をする

キーワード

▷ community.vmware.vmware_host_service_manager モジュール

方　法

ESXi の各種サービスの操作をするには community.vmware.vmware_host_service_manager を利用します。本モジュールではサービスの起動・停止やサービスポリシーの設定ができます。

List 4-12 の例では SSH 接続できるように SSH のサービスを有効化し、ESXi の起動と連動してサー

ビスが実行されるようにします。

List 4-12　community.vmware.vmware_host_service_manager モジュールの利用例: ./ESXi/esxi_services.yml

```
---
- hosts: esxi
  gather_facts: false
  tasks:
    - name: start ssh service
      community.vmware.vmware_host_service_manager:
        hostname: "{{ inventory_hostname }}"
        username: "{{ esxi_username }}"
        password: "{{ esxi_password }}"
        validate_certs: false
        esxi_hostname: "{{ inventory_hostname }}"
        service_name: TSM-SSH
        service_policy: true
        state: present
```

解　説

community.vmware.vmware_host_service_manager モジュールの主なパラメータは Table 4-8 のとおりです。

Table 4-8　community.vmware.vmware_host_service_manager モジュールの主なパラメータ

パラメータ名	説明
esxi_hostname	ESXi ホスト名を指定する
service_name	サービスの名前を指定する
service_policy	サービスの起動ポリシーを指定する
state	サービスの有効化（present）、無効化（absent）、再起動（restart）を指定する

service_policy パラメータに true を指定すると ESXi 起動時にサービスを開始します。automatic を指定するとファイアウォールでポートが開いている場合にサービスを起動します。

関　連

▷ community.vmware.vmware_host_service_manager モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_host_service_manager_module.html

4-2-5　標準仮想スイッチを作成する

キーワード

▷ community.vmware.vmware_vswitch モジュール

方　法

標準仮想スイッチを作成するには community.vmware.vmware_vswitch モジュールを利用します。本モジュールでは標準仮想スイッチの追加・削除や各種設定ができます。

List 4-13 の例では vSwitch1 という名前の標準仮想スイッチを ESXi に追加します。

List 4-13　community.vmware.vmware_vswitch モジュール: ./ESXi/esxi_vswitch.yml

```
---
- hosts: esxi
  gather_facts: false
  tasks:
    - name: create vSwitch
      community.vmware.vmware_vswitch:
        hostname: "{{ inventory_hostname }}"
        username: "{{ esxi_username }}"
        password: "{{ esxi_password }}"
        validate_certs: false
        switch: vSwitch1
        nics:
          - vmnic1
        mtu: 1500
        state: present
```

解　説

community.vmware.vmware_vswitch モジュールの主なパラメータは Table 4-9 のとおりです。

Table 4-9　community.vmware.vmware_vmware_vswitch モジュールの主なパラメータ

パラメータ名	説明
switch	標準仮想スイッチの名前を指定する
nics	標準仮想スイッチに割り当てる対象の物理 NIC のリストを指定する
mtu	標準仮想スイッチに設定する MTU の値を指定する
state	標準仮想スイッチを追加（present）するか削除（absent）するかを指定する

関　連

▷ community.vmware.vmware_vswitch モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_vswitch_module.html

4-2-6　ポートグループを作成する

キーワード

▷ community.vmware.vmware_portgroup モジュール

方　法

標準仮想スイッチにポートグループを作成するには community.vmware.vmware_portgroup モジュールを利用します。本モジュールではポートグループの作成・削除および VLAN やセキュリティなどの各種設定ができます。

List 4-14 の例では vSwitch1 に対して VLAN ID 100 のポートグループ port01 を追加します。

List 4-14　community.vmware.vmware_portgroup モジュールの利用例: ./ESXi/esxi_vswitch_portgroup.yml

```
1: ---
2: - hosts: esxi
3:   gather_facts: false
4:   tasks:
5:     - name: add portgroup
6:       community.vmware.vmware_portgroup:
7:         hostname: "{{ inventory_hostname }}"
8:         username: "{{ esxi_username }}"
9:         password: "{{ esxi_password }}"
10:        validate_certs: false
11:        hosts: "{{ inventory_hostname }}"
12:        switch: vSwitch1
13:        portgroup: port01
14:        vlan_id: 100
15:        state: present
```

解　説

community.vmware.vmware_portgroup モジュールの主なパラメータは Table 4-10 のとおりです。

Table 4-10　community.vmware.vmware_portgroup モジュールの主なパラメータ

パラメータ名		説明
hosts		ポートグループを作成する ESXi ホストを指定する
switch		標準仮想スイッチの名前を指定する
portgroup		ポートグループ名を指定する
vlan_id		ポートグループに設定する VLAN ID を指定する
security		ポートグループのセキュリティ設定を指定する
	promiscuous_mode	無差別モードを承諾（true）するか拒否（false）するかを指定する
	forged_transmits	偽装転送を承諾（true）するか拒否（false）するかを指定する
state		ポートグループを追加（present）するか削除（absent）するかを指定する

応　用

vSphere の検証環境を準備する際に Nested ESXi で構築する場合、Nested ESXi 上に作成した仮想マシンと通信するにはポートグループのセキュリティ設定が必要です。その場合の例として List 4-15 のようにセキュリティ設定をプレイブックに追加して実行することもできます。

List 4-15　セキュリティ設定を追加した例

```
---
- hosts: esxi
  gather_facts: false
  tasks:
    - name: change security for portgroup
      community.vmware.vmware_portgroup:
        hostname: "{{ inventory_hostname }}"
        username: "{{ esxi_username }}"
        password: "{{ esxi_password }}"
        validate_certs: false
        hosts: "{{ inventory_hostname }}"
        switch: vSwitch1
        portgroup: port01
        vlan_id: 100
        security:
          promiscuous_mode: true
          forged_transmits: true
        state: present
```

関　連

▷ community.vmware.vmware_portgroup モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_portgroup_module.html

4-2-7　VMKernel を追加する

キーワード

▷ community.vmware.vmware_vmkernel モジュール

方　法

VMKernel を追加するには community.vmware.vmware_vmkernel モジュールを利用します。本モジュールでは VMKernel の追加・削除および IP アドレスやサービスの有効化・無効化ができます。

List 4-16 の例ではポートグループの port01 に対して新しい VMKernel を追加します。

List 4-16　community.vmware.vmware_vmkernel モジュールの利用例: ./ESXi/esxi_vmkernel.yml

```
---
- hosts: esxi
  gather_facts: false
  tasks:
    - name: add vmkernel
      community.vmware.vmware_vmkernel:
        hostname: "{{ inventory_hostname }}"
        username: "{{ esxi_username }}"
        password: "{{ esxi_password }}"
        validate_certs: false
        esxi_hostname: "{{ inventory_hostname }}"
        vswitch_name: vSwitch1
        portgroup_name: port01
        network:
          type: static
          ip_address: 192.168.0.162
          subnet_mask: 255.255.255.0
        enable_mgmt: true
        enable_vmotion: true
        state: present
```

解　説

community.vmware.vmware_vmkernel モジュールの主なパラメータは Table 4-11 のとおりです。

Table 4-11　community.vmware.vmware_vmkernel モジュールの主なパラメータ

パラメータ名		説明
esxi_hostname		ESXi ホスト名を指定する
vswitch_name		仮想スイッチ名を指定する
portgroup_name		ポートグループ名を指定する
network		ポートグループに設定するネットワークパラメータを指定する
	type	設定するネットワークのタイプを指定する
	ip_address	静的 IP アドレスを設定する場合は、設定する IP アドレスを指定する
	subnet_mask	静的 IP アドレスを設定する場合は、設定するサブネットマスクを指定する
enable_mgmt		管理サービスを有効にするかどうかを指定する
enable_vmotion		vMotion サービスを有効にするかどうかを指定する

type パラメータに static を指定した場合は ip_address パラメータと subnet_mask パラメータに静的 IP アドレスの設定を指定します。type パラメータに dhcp を指定した場合は静的 IP アドレスの指定は不要です。

関　連

▷ community.vmware.vmware_vmkernel モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_vmkernel_module.html

4-2-8　ESXi のローカルユーザを作成する

キーワード

▷ community.vmware.vmware_local_user_manager モジュール

方　法

ESXi のローカルユーザを作成するには community.vmware.vmware_local_user_manager モジュールを利用します。本モジュールではローカルユーザの作成や削除ができます。

List 4-17 の例では ESXi に新しいユーザとして user01 を作成します。

List 4-17　community.vmware.vmware_local_user_manager モジュールの利用例: ./ESXi/esxi_user.yml

```
---
- hosts: esxi
  gather_facts: false
  tasks:
    - name: add local user
      community.vmware.vmware_local_user_manager:
        hostname: "{{ inventory_hostname }}"
        username: "{{ esxi_username }}"
        password: "{{ esxi_password }}"
        validate_certs: false
        local_user_name: user01
        local_user_password: P@ssw0rd
        local_user_description: test user
        state: present
```

解　説

community.vmware.vmware_local_user_manager モジュールのパラメータを Table 4-12 に示します。

Table 4-12　community.vmware.vmware_local_user_manager モジュールの主なパラメータ

パラメータ名	説明
local_user_name	ユーザ名を指定する
local_user_password	設定するパスワードを指定する
local_user_description	ユーザの説明文を指定する
state	ユーザを作成（present）するか削除（absent）するかを指定する

local_user_password パラメータで設定できるパスワードの文字や長さは ESXi の PAM の pam_passwdqc モジュール設定に依存します。

関　連

▷ community.vmware.vmware_local_user_manager モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_local_user_manager_module.html

▷ pam_passwdqc モジュール設定

https://docs.vmware.com/jp/VMware-vSphere/7.0/com.vmware.vsphere.security.doc/GUID-DC96FFDB-F5F2-43EC-8C73-05ACDAE6BE43.html

4-3　vCenter 基本設定

vCenter Server を対象とした各種設定などを行うためのプレイブックを紹介します。

4-3-1　フォルダを作成する

キーワード

▷ community.vmware.vcenter_folder モジュール

方　法

vCenter Server に各オブジェクト（仮想マシン、ESXi ホスト、データストア、ネットワーク）のフォルダを作成するには community.vmware.vcenter_folder モジュールを利用します。本モジュールでは各オブジェクト用のフォルダの作成、削除ができます。

List 4-18 の例では仮想マシン用の新しいフォルダとして F0 という名前で作成します。

List 4-18　community.vmware.vcenter_folder モジュールの利用例: ./vCenter/vcenter_folder.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: create folder
      community.vmware.vcenter_folder:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter: DC01
        folder_name: F0
        folder_type: vm
        state: present
```

解　説

community.vmware.vcenter_folder モジュールの主なパラメータは Table 4-13 のとおりです。

Table 4-13　community.vmware.vcenter_folder モジュールの主なパラメータ

パラメータ名	説明
datacenter	フォルダを作成するデータセンタ名を指定する
parent_folder	階層化する場合、親となるフォルダ名を指定する
folder_name	フォルダ名を指定する
folder_type	作成するフォルダのタイプを指定する
state	フォルダを作成（present）するか削除（absent）するかを指定する

応　用

フォルダは階層構造をとることができます。もし、フォルダを階層構造化したい場合は parent_folder パラメータを使用します。parent_folder パラメータには作成するフォルダの親フォルダ名を指定します。List 4-19 は F0 フォルダの配下に F1 フォルダを作成する例です。

List 4-19　F0 フォルダの配下に F1 フォルダを作成する例

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: create folder
      community.vmware.vcenter_folder:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter: DC01
        parent_folder: F0
        folder_name: F1
        folder_type: vm
        state: present
```

関　連

▷ community.vmware.vcenter_folder モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vcenter_folder_module.html

4-3-2 リソースプールを作成する

キーワード

▷ community.vmware.vmware_resource_pool モジュール

方 法

リソースプールを作成するには community.vmware.vmware_resource_pool モジュールを利用します。本モジュールではリソースプールの作成、削除および各種設定ができます。

List 4-20 の例では Cluster01 に新しいリソースプールとして RP01 を作成し CPU とメモリの予約をします。

List 4-20 community.vmware.vmware_resource_pool モジュールの利用例:
./vCenter/vcenter_resource_pool.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: create resource pool
      community.vmware.vmware_resource_pool:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter: DC01
        cluster: Cluster01
        resource_pool: RP01
        cpu_reservation: 1000
        mem_reservation: 2048
        state: present
```

解 説

community.vmware.vmware_resource_pool パラメータは Table 4-14 のとおりです。

Table 4-14 community.vmware.vmware_resource_pool モジュールの主なパラメータ

パラメータ名	説明
datacenter	リソースプールを作成するデータセンタ名を指定する
cluster	リソースプールを作成するクラスタ名を指定する
resource_pool	リソースプール名を指定する

cpu_reservation	CPU の予約サイズを MHz 単位で指定する
mem_reservation	メモリの予約サイズを MB 単位で指定する
state	リソースプールを作成（present）するか削除（absent）するか指定する

関　連

▷ community.vmware.vmware_resource_pool モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_resource_pool_module.html

4-3-3　オブジェクトのパーミッション設定をする

キーワード

▷ community.vmware.vmware_object_role_permission モジュール

方　法

各オブジェクト（データセンタ、フォルダ、仮想マシンなど）に権限を設定するには community.vmware.vmware_object_role_permission モジュールを利用します。

List 4-21 の例では vsphere.local ドメインの user01 ユーザを管理者権限で F0 のフォルダオブジェクトに追加しています。

List 4-21　community.vmware.vmware_object_role_permission モジュールの利用例: ./vCenter/vcenter_user_permission.yml

```
1: ---
2: - hosts: vcenter
3:   gather_facts: false
4:   tasks:
5:     - name: add permission
6:       community.vmware.vmware_object_role_permission:
7:         hostname: "{{ inventory_hostname }}"
8:         username: "{{ vcenter_username }}"
9:         password: "{{ vcenter_password }}"
10:        validate_certs: false
11:        role: Admin
12:        principal: vsphere.local\user01
13:        object_name: F0
```

```
14:         object_type: Folder
15:         recursive: true
16:         state: present
```

解　説

community.vmware.vmware_object_role_permission モジュールの主なパラメータは Table 4-15 のとおりです。

Table 4-15　community.vmware.vmware_object_role_permission モジュールの主なパラメータ

パラメータ名	説明
role	ロールを指定する
principal	対象のユーザを指定する
group	対象のグループを指定する
object_name	オブジェクト名を指定する
object_type	オブジェクトのタイプを指定する
recursive	再帰的に設定するかを指定する
state	権限を追加（present）するか削除（absent）するか指定する

応　用

権限の追加はグループ単位でも指定可能です。グループで指定する場合は group パラメータを使用します。

List 4-22 の例では test-users グループを指定しています。

List 4-22　権限追加でグループを指定する例

```
 1: ---
 2: - hosts: vcenter
 3:   gather_facts: false
 4:   tasks:
 5:     - name: add permission
 6:       community.vmware.vmware_object_role_permission:
 7:         hostname: "{{ inventory_hostname }}"
 8:         username: "{{ vcenter_username }}"
 9:         password: "{{ vcenter_password }}"
10:         validate_certs: false
11:         role: Admin
12:         group: vsphere.local\test-users
13:         object_name: F0
```

```
14:         object_type: Folder
15:         recursive: true
16:         state: present
```

補　足

ユーザを指定する principal とグループを指定する group は併用できません。

関　連

▷ community.vmware.vmware_object_role_permission モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_object_role_permission_module.html

4-3-4　ESXi ホストをメンテナンスモードにする

キーワード

▷ community.vmware.vmware_maintenancemode モジュール

方　法

ESXi ホストをメンテナンスモードへ移行するには community.vmware.vmware_maintenancemode モジュールを利用します。

List 4-23 の例では ESXi ホスト esxi-001.local をメンテナンスモードに移行して、電源オフの VM はすべて他の ESXi ホストへ移動させます。

List 4-23　community.vmware.vmware_maintenancemode モジュールの利用例:
./vCenter/vcenter_maintenance_mode.yml

```
1: ---
2: - hosts: vcenter
3:   gather_facts: false
4:   tasks:
5:     - name: enter maintenance mode
6:       community.vmware.vmware_maintenancemode:
7:         hostname: "{{ inventory_hostname }}"
8:         username: "{{ vcenter_username }}"
9:         password: "{{ vcenter_password }}"
10:        validate_certs: false
```

```
11:         esxi_hostname: esxi-001.local
12:         evacuate: true
13:         state: present
```

解　説

community.vmware.vmware_maintenancemode モジュールの主なパラメータは Table 4-16 のとおりです。

Table 4-16　community.vmware.vmware_maintenancemode モジュールの主なパラメータ

パラメータ名	説明
esxi_hostname	ESXi ホスト名を指定する
evacuate	メンテナンスモード移行時に電源オフの仮想マシンを移動する（true）かしない（false）かを指定する
state	メンテナンスモードに移行（present）するか終了（absent）するかを指定する

関　連

▷ community.vmware.vmware_maintenancemode モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_maintenancemode_module.html

4-3-5　カテゴリを作成する

キーワード

▷ community.vmware.vmware_category モジュール

方　法

カテゴリを作成するには community.vmware.vmware_category モジュールを利用します。カテゴリを作成することでタグのグループ化ができます。

List 4-24 の例では新しく category_test01 という名前でカテゴリを作成し、グループ化したタグを割り当てられるオブジェクトはクラスタ、仮想マシンを対象とします。

List 4-24　community.vmware.vmware_category モジュールの利用例: ./vCenter/vcenter_category.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: add category
      community.vmware.vmware_category:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        category_name: category_test01
        category_description: test category
        category_cardinality: multiple
        associable_object_types:
          - Cluster
          - Virtual Machine
        state: present
```

解　説

community.vmware.vmware_category モジュールの主なパラメータは Table 4-17 のとおりです。

Table 4-17　community.vmware.vmware_category モジュールの主なパラメータ

パラメータ名	説明
category_name	カテゴリ名を指定する
category_description	カテゴリの説明文を指定する
category_cardinality	カテゴリでグループ化したタグをオブジェクトに割り当てる場合に 1 つのみまたは複数設定できるようにするかどうかを指定する
associable_object_types	カテゴリの対象オブジェクトのタイプをリストで指定する
state	カテゴリを追加（present）するか削除（absent）するかを指定する

応　用

作成したカテゴリに対してタグを作成するときにはカテゴリの ID が必要になります。カテゴリの ID を取得するには community.vmware.vmware_category_info モジュールを利用します。

List 4-25 の例ではすべてのカテゴリ情報を取得して表示します。

List 4-25　community.vmware.vmware_category_info モジュールの利用例:
./vCenter/vcenter_category_info.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: gather categories
      community.vmware.vmware_category_info:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
      register: gather_categories_result

    - name: display categories
      debug:
        msg: "{{ gather_categories_result }}"
```

List 4-26　カテゴリ情報の例

```
ok: [vcenter-coock-book.local] => {
... ( 略 ) ...
        "tag_category_info": [
            {
                "category_associable_types": [
                    "VirtualMachine",
                    "ClusterComputeResource"
                ],
                "category_cardinality": "MULTIPLE",
                "category_description": "test category",
                "category_id": "urn:vmomi:InventoryServiceCategory:e68502c5-65aa-41d3-9576
-c45d76a63889:GLOBAL",
                "category_name": "category_test01",
                "category_used_by": []
            }
        ]
... ( 略 ) ...
```

必要なカテゴリの ID は category_id の値です。こちらを使用して対象のカテゴリにタグを作成できます。

関　連

▷ community.vmware.vmware_category モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_category_module.html

▷ community.vmware.vmware_category_info モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_category_info_module.html

4-3-6 タグを作成する

キーワード

▷ community.vmware.vmware_tag モジュール

方 法

タグを作成するには community.vmware.vmware_tag モジュールを利用します。

List 4-27 の例では「4-3-14 カテゴリを作成する」で作成したカテゴリに対して新しく test_tag01 という名前のタグを作成します。

List 4-27 community.vmware.vmware_tag モジュールの利用例: ./vCenter/vcenter_tag.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: add tag
      community.vmware.vmware_tag:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        tag_name: test_tag01
        tag_description: test tag
        category_id: 'urn:vmomi:InventoryServiceCategory:e68502c5-65aa-41d3-9576⇒
-c45d76a63889:GLOBAL'
        state: present
```

解 説

community.vmware.vmware_tag モジュールの主なパラメータは Table 4-18 のとおりです。

Table 4-18　community.vmware.vmware_tag モジュールの主なパラメータ

パラメータ名	説明
tag_name	タグの名前を指定する
tag_description	タグの説明文を指定する
category_id	タグを作成するカテゴリの ID を指定する
state	タグを作成（present）するか削除（absent）するか指定する

関　連

▷ 4-3-14 カテゴリを作成する

▷ community.vmware.vmware_tag モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_tag_module.html

4-3-7　タグをオブジェクトに割り当てる

キーワード

▷ community.vmware.vmware_tag_manager モジュール

方　法

タグをオブジェクトに対して割り当てるには community.vmware.vmware_tag_manager モジュールを利用します。

List 4-28 の例では仮想マシン new-vm01 に対して test_tag01 タグを割り当てます。

List 4-28　community.vmware.vmware_tag_manager モジュールの利用例: ./vCenter/vcenter_tag_manager.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: assign tag to virtual machine
      community.vmware.vmware_tag_manager:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
```

```
11:         object_name: new-vm01
12:         object_type: VirtualMachine
13:         tag_names:
14:           - test_tag01
15:         state: present
```

解　説

community.vmware.vmware_tag_manager モジュールの主なパラメータは Table 4-19 のとおりです。

Table 4-19　community.vmware.vmware_tag_manager モジュールの主なパラメータ

パラメータ名	説明
object_name	オブジェクト名を指定する
object_type	オブジェクトのタイプを指定する
tag_names	タグをリストで指定する
state	タグを割り当てる（present、add）か削除（absent、remove）または更新（set）するかを指定する

関　連

▷ community.vmware.vmware_tag_manager モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_tag_manager_module.html

4-3-8　ライセンスを登録する

キーワード

▷ community.vmware.vcenter_license モジュール

方　法

ライセンスを登録するには community.vmware.vcenter_license モジュールを利用します。本モジュールでは vCenter Server や ESXi のライセンスを vCenter Server に登録、削除できます。

List 4-29 の例では vCenter Server と ESXi のライセンスを登録します。

List 4-29　community.vmware.vcenter_license モジュールの利用例（vCenter Server ライセンス）：./vCenter/vcenter_assign_license.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: assign license for VCSA
      community.vmware.vcenter_license:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        license: "{{ vcenter_license }}"
        state: present
```

List 4-30　community.vmware.vcenter_license モジュールの利用例（ESXi ライセンス）：./vCenter/vcenter_esxi_assign_license.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: assign license for ESXi
      community.vmware.vcenter_license:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        license: "{{ esxi_license }}"
        state: present
```

解　説

community.vmware.vcenter_license の主なパラメータは Table 4-20 のとおりです。

Table 4-20　community.vmware.vcenter_license モジュールの主なパラメータ

パラメータ名	説明
license	vCenter Server または ESXi のライセンスを指定する
state	ライセンスを登録（present）するか削除（absent）するかを指定する

関　連

▷ community.vmware.vcenter_license モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vcenter_license_module.html

4-3-9　データセンタを作成する

キーワード

▷ community.vmware.vmware_datacenter モジュール

方　法

データセンタを作成するには community.vmware.vmware_datacenter モジュールを利用します。本モジュールではデータセンタの作成、削除ができます。

List 4-31 の例では新しいデータセンタとして DC01 を作成します。

List 4-31　community.vmware.vmware_datacenter モジュールの利用例: ./vCenter/vcenter_datacenter.yml

```
1: ---
2: - hosts: vcenter
3:   gather_facts: false
4:   tasks:
5:     - name: create datacenter
6:       community.vmware.vmware_datacenter:
7:         hostname: "{{ inventory_hostname }}"
8:         username: "{{ vcenter_username }}"
9:         password: "{{ vcenter_password }}"
10:        validate_certs: false
11:        datacenter_name: DC01
12:        state: present
```

解　説

community.vmware.vmware_datacenter モジュールの主なパラメータは Table 4-21 のとおりです。

Table 4-21　community.vmware.vmware_datacenter モジュールの主なパラメータ

パラメータ名	説明
datacenter_name	データセンタ名を指定する
state	データセンタを作成（present）するか削除（absent）するかを指定する

関　連

▷ community.vmware.vmware_datacenter モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_datacenter_module.html

4-3-10　クラスタを作成する

キーワード

▷ community.vmware.vmware_cluster モジュール

方　法

クラスタを作成するには community.vmware.vmware_cluster モジュールを利用します。本モジュールではクラスタの作成、削除および各種設定ができます。

List 4-32 の例では新しいクラスタとして Cluster01 を作成します。

List 4-32　community.vmware.vmware_cluster モジュールの利用例: ./vCenter/vcenter_cluster.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: create cluster
      community.vmware.vmware_cluster:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter_name: DC01
        cluster_name: Cluster01
        state: present
```

解　説

community.vmware.vmware_cluster モジュールの主なパラメータは Table 4-22 のとおりです。

Table 4-22　community.vmware.vmware_cluster モジュールの主なパラメータ

パラメータ名	説明
datacenter_name	クラスタを作成するデータセンタを指定する

cluster_name	クラスタ名を指定する
state	クラスタを作成（present）するか削除（absent）するかを指定する

本モジュールでも DRS（Destributed Resource Schedule）や HA（vSphere High Availability）の設定が可能ですが、将来的には別のモジュールに置き換わります。DRS や HA の設定方法については推奨モジュールの節「4-2-20 DRS（Destributed Resource Scheduler）の設定をする」および「4-3-21 HA（vSphere High Availability）の設定をする」で説明します。

関　連

▷ community.vmware.vmware_cluster モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_cluster_module.html

4-3-11　DRS（Distributed Resource Scheduler）の設定をする

キーワード

▷ community.vmware.vmware_cluster_drs モジュール

方　法

クラスタの DRS を設定するには community.vmware.vmware_cluster_drs モジュールを利用します。本モジュールでは DRS の有効、無効および各種設定ができます。

List 4-33 の例では Cluster01 の DRS を有効化します。

List 4-33　community.vmware.vmware_cluster_drs モジュールの利用例: ./vCenter/vcenter_cluster_drs.yml

```
1: ---
2: - hosts: vcenter
3:   gather_facts: false
4:   tasks:
5:     - name: enable DRS
6:       community.vmware.vmware_cluster_drs:
7:         hostname: "{{ inventory_hostname }}"
8:         username: "{{ vcenter_username }}"
9:         password: "{{ vcenter_password }}"
10:        validate_certs: false
```

```
11:         datacenter_name: DC01
12:         cluster_name: Cluster01
13:         enable_drs: true
14:         drs_default_vm_behavior: fullyAutomated
15:         drs_enable_vm_behavior_overrides: true
16:         drs_vmotion_rate: 1
```

解　説

community.vmware.vmware_cluster_drs モジュールの主なパラメータは Table 4-23 のとおりです。

Table 4-23　community.vmware.vmware_cluster_drs モジュールの主なパラメータ

パラメータ名	説明
datacenter_name	クラスタが存在するデータセンタを指定する
cluster_name	クラスタ名を指定する
enable_drs	DRS を有効にするかどうかを指定する
drs_default_vm_behavior	DRS の自動化レベルを指定する
drs_enable_vm_behavior_overrides	個々の仮想マシンで設定した DRS の動作を上書きするかどうかを指定する
drs_vmotion_rate	vMotion で移行する閾値を指定する

DRS の設定は community.vmware.vmware_cluster モジュールではなく、本モジュールを使用することが推奨されています。

関　連

▷ community.vmware.vmware_cluster_drs モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_cluster_drs_module.html

4-3-12　HA（vSphere High Availability）の設定をする

キーワード

▷ community.vmware.vmware_cluster_ha モジュール

方　法

クラスタの HA 設定をするには community.vmware.vmware_cluster_ha モジュールを利用します。本

モジュールでは HA の有効、無効および各種設定ができます。

List 4-34 の例では Cluster01 の HA の有効化します。

List 4-34　community.vmware.vmware_cluster_ha モジュールの利用例: ./vCenter/vcenter_cluster_ha.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: enable HA
      community.vmware.vmware_cluster_ha:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter_name: DC01
        cluster_name: Cluster01
        enable_ha: true
        ha_host_monitoring: enabled
        ha_restart_priority: medium
        ha_vm_monitoring: vmMonitoringOnly
        ha_vm_failure_interval: 30
        ha_vm_min_up_time: 120
        ha_vm_max_failures: 3
        ha_vm_max_failure_window: -1
```

解　説

community.vmware.vmware_cluster_ha モジュールの主なパラメータは Table 4-24 のとおりです。

Table 4-24　community.vmware.vmware_cluster_ha モジュールの主なパラメータ

パラメータ名	説明
datacenter_name	クラスタが存在するデータセンタを指定する
cluster_name	クラスタ名を指定する
enable_ha	HA を有効にするかどうかを指定する
ha_host_monitoring	ESXi ホストの監視を有効化するかどうかを指定する
ha_restart_priority	仮想マシン再起動のデフォルトの優先度を指定する
ha_vm_monitoring	仮想マシン監視を有効にするかどうかを指定する
ha_vm_failure_interval	仮想マシン監視の障害間隔を指定する
ha_vm_min_up_time	仮想マシン監視のアップタイム最小値を指定する
ha_vm_max_failures	仮想マシン監視の最大リセット回数を指定する

ha_vm_max_failure_window	仮想マシン監視の最大リセット回数に設定された回数分のリセットを実行する範囲時間（単位は秒）

HA の設定は community.vmware.vmware_cluster モジュールではなく、本モジュールを使用することが推奨されています。

関　連

▷ community.vmware.vmware_cluster_ha モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_cluster_ha_module.html

4-3-13　ESXi ホストを追加する

キーワード

▷ community.vmware.vmware_host モジュール

方　法

ESXi ホストを vCenter Server に追加するには community.vmware.vmware_host モジュールを利用します。

List 4-35 の例では esxi-001.local を Cluster01 に追加します。

List 4-35　community.vmware.vmware_host モジュールの利用例: ./vCenter/vcenter_esxi.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: add ESXi host
      community.vmware.vmware_host:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter: DC01
        cluster: Cluster01
        esxi_hostname: esxi-001.local
```

```
14:         esxi_username: root
15:         esxi_password: "{{ esxi_password }}"
16:         state: present
```

解　説

community.vmware.vmware_host モジュールの主なパラメータは Table 4-25 のとおりです。

Table 4-25　community.vmware.vmware_host モジュールの主なパラメータ

パラメータ名	説明
datacenter	ESXi ホストを追加するデータセンタ名を指定する
cluster	ESXi ホストを追加するクラスタ名を指定する
esxi_hostname	追加する ESXi ホスト名を指定する
esxi_username	追加する ESXi ホストの管理者ユーザ名を指定する
esxi_password	追加する ESXi ホストの管理者ユーザのパスワードを指定する
state	ESXi ホストの状態

state パラメータに指定する ESXi ホストの状態は Table 4-26 のとおりです。

Table 4-26　state パラメータに指定する ESXi ホストの状態

state パラメータの値	説明
present	ESXi ホストを追加
absent	ESXi ホストを削除
add_or_reconnect	ESXi ホストを追加または再接続
reconnect	ESXi ホストを再接続
disconnected	ESXi ホストを切断

補　足

本モジュールで ESXi ホストを追加しただけでは vCenter Server に登録したライセンスの紐付けがされません。そのため、community.vmware.vcenter_license モジュールを利用して登録した ESXi ホストに対してライセンスを紐付けする必要があります。

List 4-36 は community.vmware.vcenter_license モジュールで紹介した ESXi ライセンス登録のプレイブックに、esxi_hostname パラメータを追加したものです。esxi_hostname パラメータにはライセンスを紐付ける ESXi ホスト名を指定します。

List 4-36　community.vmware.vcenter_license モジュールを利用して追加した ESXi ホストにライセンスを紐付ける例

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: assign license for ESXi
      community.vmware.vcenter_license:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        license: "{{ esxi_license }}"
        esxi_hostname: esxi-001.local
        state: present
```

関　連

▷ community.vmware.vmware_host モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_host_module.html

4-3-14　分散仮想スイッチを作成する

キーワード

▷ community.vmware.vmware_dvswitch モジュール

方　法

分散仮想スイッチを作成するには community.vmware.vmware_dvswitch モジュールを利用します。本モジュールでは仮想スイッチの作成、削除および各種設定ができます。

List 4-37 の例では新しい分散仮想スイッチとして dvswitch01 を作成します。

List 4-37　community.vmware.vmware_dvswitch モジュールの利用例: ./vCenter/vcenter_dvswitch.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: create dvSwitch
```

```
 6:      community.vmware.vmware_dvswitch:
 7:        hostname: "{{ inventory_hostname }}"
 8:        username: "{{ vcenter_username }}"
 9:        password: "{{ vcenter_password }}"
10:        validate_certs: false
11:        datacenter: DC01
12:        switch_name: dvswitch01
13:        version: 7.0.0
14:        mtu: 1500
15:        uplink_quantity: 2
16:        state: present
```

解　説

community.vmware.vmware_dvswitch パラメータは Table 4-27 のとおりです。

Table 4-27　community.vmware.vmware_dvswitch モジュールの主なパラメータ

パラメータ名	説明
datacenter	分散仮想スイッチを作成するデータセンタ名を指定する
switch_name	分散仮想スイッチ名を指定する
version	分散仮想スイッチのバージョンを指定する
mtu	分散仮想スイッチに設定する MTU 値を指定する
uplink_quantity	分散仮想スイッチに設定するアップリンク数を指定する
state	分散仮想スイッチを作成（state）するか削除（absent）するかを指定する

関　連

▷ community.vmware.vmware_dvswitch モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_dvswitch_module.html

4-3-15　分散仮想スイッチに ESXi ホストを追加する

キーワード

▷ community.vmware.vmware_dvs_host モジュール

方　法

分散仮想スイッチに ESXi ホストを追加するには community.vmware.vmware_dvs_host モジュールを利用します。本モジュールではアップリンクの登録や LACP(Link Aggregation Control Protocol）の設定もできます。

List 4-38 の例では分散仮想スイッチの dvswitch01 に ESXi ホストの esxi-001.local を追加します。

List 4-38　community.vmware.vmware_dvs_host モジュールの利用例: ./vCenter/vcenter_dvs_host.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: add ESXi host to dvs
      community.vmware.vmware_dvs_host:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        switch_name: dvswitch01
        esxi_hostname: esxi-001.local
        vmnics:
          - vmnic2
          - vmnic3
        state: present
```

解　説

community.vmware.vmware_dvs_host モジュールの主なパラメータは Table 4-28 のとおりです。

Table 4-28　community.vmware.vmware_dvs_host モジュールの主なパラメータ

パラメータ名	説明
switch_name	ESXi ホストを追加する分散仮想スイッチ名を指定する
esxi_hostname	ESXi ホスト名を指定する
vmnics	仮想分散スイッチに使用する ESXi ホストのアップリンクを指定する

関　連

▷ community.vmware.vmware_dvs_host モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_dvs_host_module.html

4-3-16 分散ポートグループを作成する

キーワード

▷ community.vmware.vmware_dvs_portgroup モジュール

方 法

分散ポートグループを作成するには community.vmware.vmware_dvs_portgroup モジュールを利用します。本モジュールでは分散ポートグループの作成、削除および各種設定ができます。

List 4-39 の例では分散仮想スイッチの dvswitch01 に VLAN ID 100 の分散ポートグループ dvpg01 を作成します。

List 4-39 community.vmware.vmware_dvs_portgroup モジュールの利用例:
./vCenter/vcenter_dvs_portgroup.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: create dvs portgroup
      community.vmware.vmware_dvs_portgroup:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        switch_name: dvswitch01
        portgroup_name: dvpg01
        vlan_id: 100
        portgroup_type: earlyBinding
        state: present
```

解 説

community.vmware.vmware_dvs_portgroup モジュールの主なパラメータは Table 4-29 のとおりです。

Table 4-29 community.vmware.vmware_dvs_portgroup モジュールの主なパラメータ

パラメータ名	説明
switch_name	分散ポートグループを追加する分散仮想スイッチ名を指定する
portgroup_name	分散ポートグループ名を指定する
vlan_id	分散ポートグループに設定する VLAN ID を指定する（VLAN を使用しない場合は 0 を指定する）

vlan_trunk	分散ポートグループを VLAN のトランクポートとして使用するかどうかを指定する
portgroup_type	分散ポートグループのポートバインドのタイプを指定する
state	分散ポートグループを作成（present）するか削除（absent）するかを指定する

応　用

分散ポートグループではトランクポートを使用できます。トランクポートで VLAN ID を設定する場合、ハイフンを使った範囲指定、またはカンマ区切りで複数指定できます。

List 4-40 は分散ポートグループの VLAN 設定を VLAN トランクにして、いくつかの VLAN ID を追加する例です。

List 4-40　VLAN トランクの設定をする例

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: add dvs portgroup
      community.vmware.vmware_dvs_portgroup:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        switch_name: dvswitch01
        portgroup_name: dvpg01
        vlan_id: 100-200, 300, 350
        vlan_trunk: true
        portgroup_type: earlyBinding
        state: present
```

関　連

▷ community.vmware.vmware_dvs_portgroup モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_dvs_portgroup_module.html

4-4　仮想マシン操作

仮想マシンのデプロイや複製、各種設定、ゲスト OS の操作などを行うためのプレイブックを紹介します。

4-4-1　仮想マシンを作成する

キーワード

▷ community.vmware.vmware_guest モジュール

方　法

仮想マシンを新規に作成するには community.vmware.vmware_guest モジュールを利用します。本モジュールでは仮想マシンの作成、削除および各種設定ができます。

List 4-41 の例では新規の仮想マシンとして new-vm01 を作成し起動します。

List 4-41　community.vmware.vmware_guest モジュールの利用例: ./VM/vm_create.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: create vm
      community.vmware.vmware_guest:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter: DC01
        cluster: Cluster01
        folder: /vm/F0
        name: new-vm01
        guest_id: centos8_64Guest
        hardware:
          num_cpus: 2
          num_cpu_cores_per_socket: 2
          memory_mb: 1024
        cdrom:
          - controller_number: 0
            unit_number: 0
            iso_path: "[nfs_ds01] ISO/CentOS-8.3.2011-x86_64-minimal.iso"
```

```
24:             type: iso
25:         disk:
26:           - size_gb: 32
27:             type: thin
28:             datastore: nfs_ds01
29:         networks:
30:           - name: VM Network
31:         state: poweredon
```

解　説

community.vmware.vmware_guest モジュールの主なパラメータは Table 4-30 のとおりです。

Table 4-30　community.vmware.vmware_guest モジュールの主なパラメータ

パラメータ名		説明
datacenter		仮想マシンを作成するデータセンタ名を指定する
cluster		仮想マシンを作成するクラスタ名を指定する
folder		仮想マシンを作成するフォルダ名を指定する
template		仮想マシンを複製するテンプレートを指定する
name		仮想マシン名を指定する
guest_id		仮想マシンで使用するゲスト OS の識別子を指定する
hardware		仮想マシンに設定するハードウェアを指定する
	num_cpus	CPU のコア数を指定する
	num_cpu_cores_per_socket	ソケットあたりのコア数を指定する
	memory_mb	メモリサイズ（MB）を指定する
cdrom		仮想マシンに設定する仮想 CD/DVD ドライブの設定を指定する
	controller_number	仮想デバイスノードのコントローラ番号を指定する
	unit_number	仮想デバイスノードのユニット番号を指定する
	iso_path	マウントする ISO イメージファイルのパスを指定する
	type	仮想 CD/DVD でマウントするメディアのタイプを指定する
disk		仮想マシンに設定する仮想ディスクを指定する
	size_gb	仮想ディスクのサイズ（GB）を指定する
	type	仮想ディスクのタイプを指定する
	datastore	仮想ディスクを作成するデータストア名を指定する
networks		仮想マシンに設定するネットワークを指定する
	name	接続するポートグループ名を指定する
	ip	設定する IP アドレスを指定する
	netmask	設定するネットマスクを指定する

	gateway	設定するゲートウェイを指定する
state		仮想マシンを作成（present）、削除（absent）または電源関連（poweredon など）の操作を指定する
customization		仮想マシンのカスタマイズを指定する
	hostname	設定するホスト名を指定する
	timezone	設定するタイムゾーンを指定する
	dns_servers	設定する DNS サーバを指定する
wait_for_customization		カスタマイズが完了するまで待つかどうかを指定する

新規で仮想マシンを作成するときには guest_id パラメータにゲスト OS の識別子が必要です。指定可能なゲスト OS の識別子は vSphere Web Services API ドキュメントの VirtualMachineGuestOsIdentifier[*7] から確認できます。

応　用

本モジュールでは template パラメータに複製元となる仮想マシンまたはテンプレートを指定して、仮想マシンの複製もできます。

List 4-42 の例では仮想マシンの new-vm01 から複製して、新しく clone-vm01 という仮想マシンを作成します。また、customization パラメータを指定することでゲスト OS のホスト名やタイムゾーンなども自動的に設定できます。

List 4-42　community.vmware.vmware_guest モジュールの利用例: ./VM/vm_clone.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: clone vm
      community.vmware.vmware_guest:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
```

＊7　VirtualMachineGuestOsIdentifier（vSphere 7.0 のゲスト OS 識別子一覧）
https://vdc-download.vmware.com/vmwb-repository/dcr-public/b50dcbbf-051d-4204-a3e7-e1b618c1e384/538cf2ec-b34f-4bae-a332-3820ef9e7773/vim.vm.GuestOsDescriptor.GuestOsIdentifier.html

```
11:         datacenter: DC01
12:         cluster: Cluster01
13:         folder: /vm/F0
14:         template: new-vm01
15:         name: clone-vm01
16:         hardware:
17:           num_cpus: 2
18:           memory_mb: 1024
19:         networks:
20:           - name: VM Network
21:             ip: 192.168.10.2
22:             netmask: 255.255.255.0
23:             gateway: 192.168.10.1
24:         customization:
25:           hostname: clone-vm01
26:           timezone: Asia/Tokyo
27:           dns_servers:
28:             - 8.8.8.8
29:         wait_for_customization: true
30:         state: poweredon
```

補　足

本モジュールでは新規に作成した仮想マシンに対して、ゲスト OS を自動でインストールすることはできません。たとえば、ゲスト OS として Linux のインストールも自動化する場合は Kickstart などの別の仕組みと組み合わせて作業を実施してください。

関　連

▷ community.vmware.vmware_guest モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_guest_module.html

4-4-2　OVF からデプロイする

キーワード

▷ community.vmware.vmware_deploy_ovf モジュール

方　法

OVF（Open Virtualization Format）から仮想マシンをデプロイするには community.vmware.vmware_deploy_ovf モジュールを利用します。

List 4-43 の例では OVF から新しい仮想マシンとして ovf-vm01 をデプロイします。

List 4-43　community.vmware.vmware_deploy_ovf モジュールの利用例: ./VM/vm_deploy_ovf.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: deploy vm with ovf file
      community.vmware.vmware_deploy_ovf:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter: DC01
        cluster: Cluster01
        name: ovf-vm01
        datastore: nfs_ds01
        networks:
          "VM Network": dvpg01
        ovf: "{{ inventory_dir }}/export_ovf/CentOS7_TMP/CentOS7_TMP.ovf"
        power_on: false
```

解　説

community.vmware.vmware_deploy_ovf モジュールの主なパラメータは Table 4-31 のとおりです。

Table 4-31　community.vmware.vmware_deploy_ovf モジュールの主なパラメータ

パラメータ名	説明
datacenter	仮想マシンを作成するデータセンタ名を指定する
cluster	仮想マシンを作成するクラスタ名を指定する
name	仮想マシン名を指定する
datastore	仮想マシンを作成するデータストア名を指定する
networks	接続するポートグループ名を指定する
ovf	OVF ファイルのパスを指定する
power_on	デプロイ後に仮想マシンの電源を付けるかどうかを指定する

OVF からデプロイする仮想マシンのポートグループを指定する場合は、networks パラメータに OVF に設定されているポートグループ名と新しく設定するポートグループ名を紐付けます。OVF に設定さ

れているポートグループ名は、OVF としてエクスポートした仮想マシンに設定していたポートグループ名です。プレイブックの例では、OVF にはポートグループとして VM Network を設定していたため、新しく設定するポートグループの dvpg01 と紐付けています。

関　連

▷ community.vmware.vmware_deploy_ovf モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_deploy_ovf_module.html

4-4-3　OVF でエクスポートする

キーワード

▷ community.vmware.vmware_export_ovf モジュール

方　法

仮想マシンを OVF としてエクスポートするには community.vmware.vmware_export_ovf モジュールを利用します。

List 4-44 の例では仮想マシン new-vm01 をエクスポートの対象として、OVF ファイルを export_ovf ディレクトリに保存します。

List 4-44　community.vmware.vmware_export_ovf モジュールの利用例: ./VM/vm_export_ovf.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: export ovf
      community.vmware.vmware_export_ovf:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter: DC01
        folder: /vm/F0
        name: new-vm01
        export_dir: "{{ inventory_dir }}/export_ovf"
```

解　説

community.vmware.vmware_export_ovf モジュールの主なパラメータは Table 4-32 のとおりです。

Table 4-32　community.vmware.vmware_export_ovf モジュールの主なパラメータ

パラメータ名	説明
datacenter	エクスポートする仮想マシンが存在するデータセンタ名を指定する
folder	エクスポートする仮想マシンが存在するフォルダ名を指定する
name	エクスポートする仮想マシン名を指定する
export_dir	OVF ファイルの保存先パスを指定する

関　連

▷ community.vmware.vmware_export_ovf モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_export_ovf_module.html

4-4-4　スナップショットを作成する

キーワード

▷ community.vmware.vmware_guest_snapshot モジュール

方　法

仮想マシンのスナップショットの作成をするには community.vmware.vmware_guest_snapshot モジュールを利用します。本モジュールを使用することでスナップショットの作成、削除および指定したスナップショットの状態に戻すことができます。

List 4-45 の例では仮想マシン new-vm01 のスナップショットを snapshot01 という名前で作成します。

List 4-45　community.vmware.vmware_guest_snapshot モジュールの利用例: ./VM/vm_snapshot.yml

```
1: ---
2: - hosts: vcenter
3:   gather_facts: false
4:   tasks:
5:     - name: create snapshot
6:       community.vmware.vmware_guest_snapshot:
```

```
 7:         hostname: "{{ inventory_hostname }}"
 8:         username: "{{ vcenter_username }}"
 9:         password: "{{ vcenter_password }}"
10:         validate_certs: false
11:         datacenter: DC01
12:         folder: /vm/F0
13:         name: new-vm01
14:         snapshot_name: snapshot01
15:         state: present
```

解　説

community.vmware.vmware_guest_snapshot パラメータは Table 4-33 のとおりです。

Table 4-33　community.vmware.vmware_guest_snapshot モジュールの主なパラメータ

パラメータ名	説明
datacenter	仮想マシンが存在するデータセンタ名を指定する
folder	仮想マシンが存在するフォルダ名を指定する
name	スナップショットを作成する仮想マシン名を指定する
snapshot_name	スナップショット名を指定する
state	スナップショットを作成（present）、削除（absent、remove_all）するかスナップショットの状態へ戻す（revert）か指定する

応　用

仮想マシンの状態を指定したスナップショットに戻すには、state パラメータに revert を指定します。List 4-46 の例では作成した snapshot01 を元に仮想マシンの状態を戻します。

List 4-46　スナップショットを元に仮想マシンの状態を戻す

```
1: ---
2: - hosts: vcenter
3:   gather_facts: false
4:   tasks:
5:     - name: revert virtual machine from snapshot
6:       community.vmware.vmware_guest_snapshot:
7:         hostname: "{{ inventory_hostname }}"
8:         username: "{{ vcenter_username }}"
9:         password: "{{ vcenter_password }}"
10:        validate_certs: false
11:        datacenter: DC01
```

```
12:         folder: /vm/F0
13:         name: new-vm01
14:         snapshot_name: snapshot01
15:         state: revert
```

関　連

▷ community.vmware.vmware_guest_snapshot モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_guest_snapshot_module.html

4-4-5　スクリーンショットを取得する

キーワード

▷ community.vmware.vmware_guest_screenshot モジュール

方　法

仮想マシンのスクリーンショットを取得するには community.vmware.vmware_guest_screenshot モジュールを利用します。

List 4-47 の例では仮想マシン new-vm01 のスクリーンショットを取得します。取得したスクリーンショットは PNG ファイルで保存されます。

List 4-47　community.vmware.vmware_guest_screenshot モジュールの利用例: ./VM/vm_screenshot.yml

```
 1: ---
 2: - hosts: vcenter
 3:   gather_facts: false
 4:   tasks:
 5:     - name: make and fetch vm screenshot
 6:       community.vmware.vmware_guest_screenshot:
 7:         hostname: "{{ inventory_hostname }}"
 8:         username: "{{ vcenter_username }}"
 9:         password: "{{ vcenter_password }}"
10:         validate_certs: false
11:         datacenter: DC01
12:         cluster: Cluster01
13:         folder: /vm/F0
14:         name: new-vm01
```

```
15:         local_path: "{{ inventory_dir }}/screenshot"
```

解　説

community.vmware.vmware_guest_screenshot モジュールの主なパラメータは Table 4-34 のとおりです。

Table 4-34　community.vmware.vmware_guest_screenshot モジュールの主なパラメータ

パラメータ名	説明
datacenter	仮想マシンが存在するデータセンタ名を指定する
folder	仮想マシンが存在するフォルダ名を指定する
name	仮想マシン名を指定する
local_path	取得したスクリーンショットの保存先パスを指定する

補　足

取得できる仮想マシンのスクリーンショットは、ウェブまたはリモートコンソールに表示される画面です。

注意点として電源が付いていない仮想マシンに対してはスクリーンショットを取得できません。

関　連

▷ community.vmware.vmware_guest_screenshot モジュール

`https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_guest_screenshot_module.html`

4-4-6　ゲスト OS にファイルをコピーする

キーワード

▷ community.vmware.vmware_guest_file_operation モジュール

方　法

ゲスト OS のファイルをコピーするには community.vmware.vmware_guest_file_operation モジュールを利用します。本モジュールを利用することで vSphere Web Service API 経由でコントロールノードとゲスト OS 間で相互にファイルのコピーができます。

List 4-48 の例では `test.zip` ファイルを Linux ゲスト OS の new-vm01 の `tmp` ディレクトリにコピー

します。

List 4-48　community.vmware.vmware_guest_file_operation モジュールの利用例:
./VM/vm_copy_file_operation.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: copy file to vm
      community.vmware.vmware_guest_file_operation:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter: DC01
        cluster: Cluster01
        folder: /vm/F0
        vm_id: new-vm01
        vm_username: root
        vm_password: P@ssw0rd
        copy:
          src: "{{ inventory_dir }}/test.zip"
          dest: "/tmp/test.zip"
          overwrite: true
```

解　説

community.vmware.vmware_guest_file_operation モジュールの主なパラメータは Table 4-35 のとおりです。

Table 4-35　community.vmware.vmware_guest_file_operation モジュールの主なパラメータ

パラメータ名		説明
datacenter		仮想マシンが存在するデータセンタ名を指定する
cluster		仮想マシンが存在するクラスタ名を指定する
folder		仮想マシンが存在するフォルダ名を指定する
vm_id		仮想マシン名を指定する
vm_username		ゲスト OS のユーザ名を指定する
vm_password		ゲスト OS のユーザパスワードを指定する
copy		ゲスト OS へファイルをコピーするパラメータを指定する
	src	ゲスト OS へコピーするファイルのパスを指定する
	dest	ゲスト OS へコピーするファイルの保存先の絶対パスを指定する

	overwrite	すでに同じファイルが存在する場合、上書きするかどうかを指定する
fetch		ゲスト OS からファイルをコピーするパラメータを指定する
	src	ゲスト OS からコピーするファイルの絶対パスを指定する
	dest	ゲスト OS からコピーしたファイルの保存先パスを指定する

応　用

ゲスト OS 内にあるファイルをコントロールノード側へコピーする場合は fetch パラメータを使用します。

List 4-49 の例ではゲスト OS 内の/etc/hosts ファイルをコピーします。

List 4-49　Linux ゲスト OS から/etc/hosts ファイルをコピーする例: ./VM/vm_fetch_file_operation.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: fetch file from vm
      community.vmware.vmware_guest_file_operation:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter: DC01
        cluster: Cluster01
        folder: /vm/F0
        vm_id: new-vm01
        vm_username: root
        vm_password: P@ssw0rd
        fetch:
          src: "/etc/hosts"
          dest: "{{ inventory_dir }}/hosts"
```

関　連

▷ community.vmware.vmware_guest_file_operation モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_guest_file_operation_module.html

4-4-7 ゲスト OS のコマンドを実行する

キーワード

▷ community.vmware.vmware_vm_shell モジュール

方 法

ゲスト OS のコマンドを実行するには community.vmware.vmware_vm_shell モジュールを利用します。本モジュールを利用することで vSphere Web Service API 経由でゲスト OS のコマンドを実行できます。

List 4-50 の例では Linux ゲスト OS の new-vm01 で touch コマンドを実行して/tmp ディレクトリに test_file.txt を作成します。

List 4-50 community.vmware.vmware.vmware_vm_shell モジュールの利用例: ./VM/vm_shell_linux.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: execute touch command
      community.vmware.vmware_vm_shell:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter: DC01
        cluster: Cluster01
        folder: /vm/F0
        vm_id: new-vm01
        vm_username: root
        vm_password: P@ssw0rd
        vm_shell: /bin/touch
        vm_shell_args: "test_file.txt"
        vm_shell_cwd: /tmp
```

解 説

community.vmware.vmware_vm_shell モジュールの主なパラメータは Table 4-36 のとおりです。

Table 4-36 community.vmware.vmware_vm_shell モジュールの主なパラメータ

パラメータ名	説明
datacenter	仮想マシンが存在するデータセンタ名を指定する

cluster	仮想マシンが存在するクラスタ名を指定する
folder	仮想マシンが存在するフォルダ名を指定する
vm_id	仮想マシン名を指定する
vm_username	ゲスト OS のユーザ名を指定する
vm_password	ゲスト OS のユーザパスワードを指定する
vm_shell	実行するコマンドの絶対パスを指定する
vm_shell_args	コマンドの引数を指定する
vm_shell_cwd	実行するディレクトリパスを指定する

応　用

Windows ゲスト OS で PowerShell などのコマンドを実行することもできます。

List 4–51 の例では PowerShell のコマンドレットを実行して Administrator ユーザのデスクトップフォルダに test_file.txt を作成します。

List 4–51　PowerShell のコマンドレットを実行する例: ./VM/vm_shell_windows.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: execute touch command
      community.vmware.vmware_vm_shell:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter: DC01
        cluster: Cluster01
        folder: /vm/F0
        vm_id: win-vm01
        vm_username: administrator
        vm_password: P@ssw0rd
        vm_shell: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
        vm_shell_args: "New-Item -Type File test_file.txt"
        vm_shell_cwd: C:\Users\Administrator\Desktop
```

関　連

▷ community.vmware.vmware_vm_shell モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/

vmware_vm_shell_module.html

4-4-8 vMotion を実行する

キーワード

▷ community.vmware.vmware_vmotion モジュール

方 法

vMotion を実行して仮想マシンを別の ESXi ホストやデータストアへ移動するには community.vmware.vmware_vmotion モジュールを利用します。

List 4-52 の例では仮想マシン new-vm01 を ESXi ホスト esxi-002.local へ移動させます。

List 4-52 community.vmware.vmware_vmotion モジュールの利用例: ./VM/vm_vmotion.yml

```
1: ---
2: - hosts: vcenter
3:   gather_facts: false
4:   tasks:
5:     - name: vmotion
6:       community.vmware.vmware_vmotion:
7:         hostname: "{{ inventory_hostname }}"
8:         username: "{{ vcenter_username }}"
9:         password: "{{ vcenter_password }}"
10:        validate_certs: false
11:        vm_name: new-vm01
12:        destination_host: esxi-002.local
```

解 説

community.vmware.vmware_vmotion モジュールの主なパラメータは Table 4-37 のとおりです。

Table 4-37 community.vmware.vmware_vmotion モジュールの主なパラメータ

パラメータ名	説明
vm_name	仮想マシン名を指定する
destination_host	移動先の ESXi ホスト名を指定する

関 連

▷ community.vmware.vmware_vmotion モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_vmotion_module.html

4-5　基本運用

仮想化基盤（VMware vSphere）の運用には必要かと思われるログ取得やレポート作成の Tips をプレイブックにして紹介します。

4-5-1　ESXi のログを取得する

キーワード

▷ community.vmware.vmware_host_logbundle モジュール

▷ community.vmware.vmware_host_logbundle_info モジュール

方　法

ESXi ホストのシステムログを取得するには community.vmware.vmware_host_logbundle モジュールを利用します。

List 4-53 の例では ESXi ホスト esxi-001.local から manifests パラメータで指定したログを含めて取得します。

List 4-53　community.vmware.vmware_host_logbundle モジュールの利用例: ./operation_and_maintenance/esxi_logbundle.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: fetch ESXi log
      community.vmware.vmware_host_logbundle:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        esxi_hostname: esxi-001.local
        dest: "{{ inventory_dir }}/esxi_log/esxi-001.local.tgz"
        manifests:
          - System:Base
          - VirtualMachines:VirtualMachineStats
```

解　説

community.vmware.vmware_host_logbundle モジュールの主なパラメータは Table 4-38 のとおりです。

Table 4-38　community.vmware.vmware_host_logbundle モジュールの主なパラメータ

パラメータ名	説明
esxi_hostname	ログを取得する対象の ESXi ホスト名を指定する
dest	ログの保存先パスを指定する
manifests	ログに含める対象を指定する

応　用

manifests パラメータにはログに含める対象を指定します。指定できる対象について確認するには community.vmware.vmware_host_logbundle_info モジュールを利用します。

List 4-54 の例では指定可能な manifests 情報を ESXi ホストから取得します。

List 4-54　community.vmware.vmware_host_logbundle_info モジュールの利用例:
./operation_and_maintenance/esxi_logbundle_info.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: fetch manifests
      community.vmware.vmware_host_logbundle_info:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        esxi_hostname: esxi-001.local
      register: log_manifests

    - name: display log manifests
      debug:
        msg: "{{ log_manifests }}"
```

List 4-55 の id キーの値が community.vmware.vmware_host_logbundle の manifests パラメータで指定できる対象になります。

List 4-55　取得した manifests の例

```
ok: [vcenter-coock-book.local] => {
    "msg": {
        "manifests": [
            {
                "enabled": "false",
                "group": "System",
                "id": "System:AllCoreDumps",
                "name": "AllCoreDumps",
                "vmOnly": "false"
            },
            {
                "enabled": "true",
                "group": "System",
                "id": "System:Base",
                "name": "Base",
                "vmOnly": "false"
            },
... ( 略 ) ...
```

関　連

▷ community.vmware.vmware_host_logbundle モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_host_logbundle_module.html

▷ community.vmware.vmware_host_logbundle_info モジュール

https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_host_logbundle_info_module.html

4-5-2　VM 情報のレポートを作成する

キーワード

▷ community.vmware.vmware_guest_info モジュール

方　法

仮想マシンの情報を取得する community.vmware.vmware_guest_info モジュールと ansible.builtin.template モジュールを組み合わせることで仮想マシンのサマリレポートを作成できます。本番環境に対して設定変更の自動化を適用するハードルが高い場合は、まずは情報を取得するモジュールを利用するところから始めるのもよいでしょう。

List 4-56 の例では仮想マシンのサマリ情報から Jinja2 テンプレートで作成した vmware_vm_summary_report_csv.j2 を元に CSV 形式のレポートを作成します。

List 4-56　community.vmware.vmware_guest_info モジュールを利用した仮想マシンのレポートを生成する例: ./operation_and_maintenance/vm_generate_report.yml

```
---
- hosts: vcenter
  gather_facts: false
  tasks:
    - name: gather vm summary
      community.vmware.vmware_guest_info:
        hostname: "{{ inventory_hostname }}"
        username: "{{ vcenter_username }}"
        password: "{{ vcenter_password }}"
        validate_certs: false
        datacenter: DC01
        name: new-vm01
      register: summary

    - name: generate summary report
      ansible.builtin.template:
        src: "{{ playbook_dir }}/templates/vmware_vm_summary_report_csv.j2"
        dest: "{{ inventory_dir }}/vmware_vm_summary_report.csv"
        mode: 0644
```

List 4-57 の Jinja2 テンプレートでは仮想マシンの名前、CPU 数、メモリサイズ（MB）、VMware Tools のバージョンを記載したレポートを生成します。

List 4-57　レポート生成用の Jinja2 テンプレート例: ./operation_and_maintenance/templates/vmware_vm_summary_report_csv.j2

```
name,cpu_num,mem_mb,tool_version
{{ summary.instance.hw_name }},{{ summary.instance.hw_processor_count }},⇒
{{ summary.instance.hw_memtotal_mb }},{{ summary.instance.guest_tools_version }}
```

解　説

community.vmware.vmware_guest_info モジュールの主なパラメータは Table 4-39 のとおりです。

Table 4-39　community.vmware.vmware_guest_info モジュールの主なパラメータ

パラメータ名	説明
datacenter	仮想マシンが存在するデータセンタ名を指定する
name	仮想マシン名を指定する

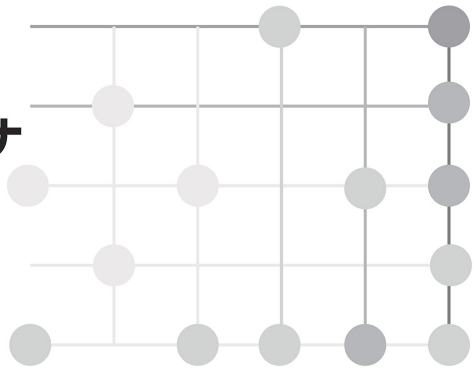
community.vmware.vmware_guest_info モジュールで取得した new-vm01 のサマリ情報を summary 変数に格納しています。summary 変数をもとに template モジュールで指定した Jinja2 テンプレートで summary 変数が展開されて CSV ファイルでレポートが生成されます。Jinja2 テンプレートの形式を変更することで別のフォーマット（HTML や Markdown など）でレポートを生成することも可能です。

関　連

▷ community.vmware.vmware_guest_info モジュール

`https://docs.ansible.com/ansible/latest/collections/community/vmware/vmware_guest_info_module.html`

第５章
コンテナ



近年のクラウドネイティブにおけるアプリケーション開発やインフラ構築において、コンテナ環境の利用頻度は大変高くなってきました。コンテナを取り巻くエコシステムも豊富で、ビルドやデプロイを行うさまざまなプロダクトもリリースされています。

Ansible にもコンテナに対応したモジュールがコレクションで提供されています。コンテナ関連については pip で ansible をインストールすると Docker、Kubernetes、Podman のコレクションがインストールされます。これらのコレクションを使うことでアプリケーションコンテナをビルドしたり、コンテナ形式で配布されているアプリケーションをデプロイできます。本章では Docker と Kubernetes を対象に、Ansible を使った自動化について紹介します。

5-1　概要

各プレイブックの紹介に先立ち、本章の前提となる環境や使用コレクションなどについて説明します（Table 5-1）。また、コネクションプラグインやコンテナ環境への接続パターンについても本節で解説します。システム構成によって是非活用してください。

Table 5-1　コンテナを扱うコレクション一覧

自動化対象	コレクション名
Docker	community.docker
Kubernetes	kubernetes.core
Podman	containers.podman

5-1-1　環境

本章では以下の環境で確認を行っています。コントロールノードは RHEL 8.4 です。

- Docker バージョン 20.10.6（Fedora 34 へインストール）
- Kubernetes バージョン 1.21.1（上記 Docker 環境へ kind v0.11.1 で構築[*1]）

また、Kubernetes では Helm を使ったアプリケーションのデプロイも行います。Helm のバージョンは v3.6.3 を使用しています。

5-1-2　使用コレクションとバージョン

使用したコレクションとバージョンは Ansible 4.2.0 に含まれているのは Table 5-2 のとおりです。

Table 5-2　本章で使用するコレクションとバージョン一覧

コレクション	バージョン
community.docker	1.8.0
kubernetes.core	1.2.1

なお、本章で使用する各コレクションおよびモジュールの実行には Table 5-3 に示した Python パッケージがマネージドノードに必要です。

＊1　https://kind.sigs.k8s.io/

Table 5-3　コンテナを扱うコレクションが依存する Python パッケージとバージョン

コレクション	必要パッケージ	本章で確認したバージョン
community.docker	docker	5.0.0
kubernetes.core	openshift	0.12.0

Column　Kubernetes のコレクション名

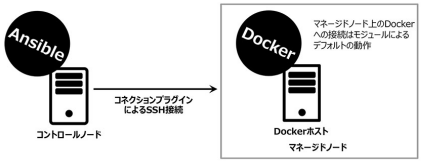
Kubernetes を管理するためのコレクションは当初は「community.kubernetes」という名前で開発されていました[*2]。現在の Ansible 4 には両方のコレクションが同梱されますが、「community.kubernetes」は非推奨（Ansible 6 で廃止予定）で「kubernetes.core」に統一される予定です[*3][*4]。community.kubernetes を使用している場合は kubernetes.core へ更新することを検討してください。

5-1-3　接続方式

■ Docker

本章では Fedora 34 にインストールした Docker 20.10.6 を対象に動作検証しています。マネージドノードに Docker が動作している Fedora ホストを指定して Linux の自動化同様 SSH コネクションでマネージドノードへ接続し、マネージドノード上で Docker 操作を行う構成です（Figure 5-1）。

Figure 5-1　マネージドノードの Docker を操作



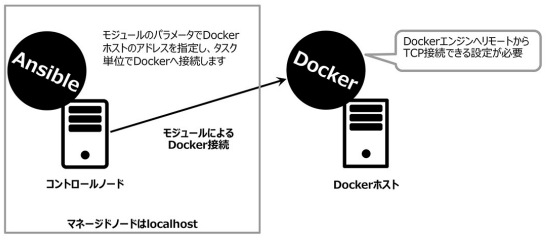
＊2　community.kubernetes
https://galaxy.ansible.com/community/kubernetes

＊3　Moving this content to kubernetes.core (v2.0)・Issue #32・ansible-collections/kubernetes.core
https://github.com/ansible-collections/kubernetes.core/issues/32

＊4　Deprecation and removal schedule for community.kubernetes・Issue #22・ansible-community/community-topics
https://github.com/ansible-community/community-topics/issues/22

コントロールノードから直接 Docker ホストを指定することもできます（Figure 5-2）。接続先に localhost を指定し、使用する各モジュールで共通の docker_host パラメータにリモートの Docker ホストを指定します（List 5-1）。この場合は対象となる Docker ホストはリモートからの接続を許可するために TCP ソケットの有効化やファイアウォールの許可設定が必要です。リモート接続についての詳細は Docker のドキュメントを参照してください[*5]。

Figure 5-2　コントロールノードからリモートの Docker を直接指定



List 5-1　docker_host を使ったリモートの Docker ホスト指定

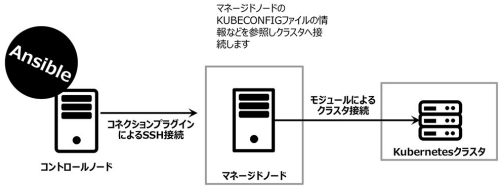
```
1: ---
2: - hosts: localhost
3:   gather_facts: false
4:
5:   tasks:
6:     - name: pull image
7:       community.docker.docker_image:
8:         name: "centos:8"
9:         source: pull
10:        docker_host: tcp://192.168.0.44:2375
```

＊5　リモート接続の詳細
https://docs.docker.com/engine/reference/commandline/dockerd/#daemon-socket-option

■ Kubernetes

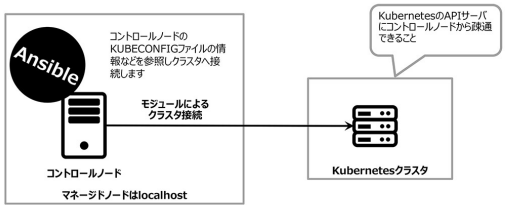
本書では kind を使用した Kubernetes クラスタを対象に動作検証しています（Figure 5-3）。このクラスタは Docker の節同様、Fedora 34 にインストールした Docker で動作しています。kind のバージョンは v0.11.1 で Kubernetes バージョンは 1.21.1 です。マネージドノードに Docker および kind の Kubernetes クラスタが動作している Fedora ホストを指定しています。

Figure 5-3　マネージドノードからクラスタ接続



コントロールノードから直接 Kubernetes クラスタへ疎通可能で、クラスタ接続のための KUBECONFIG ファイルが Ansible 実行ユーザに設定済みであれば、接続先に localhost を指定してコントロールノードから直接クラスタへアクセスできます（Figure 5-4）。その場合はコントロールノードとマネージドノードが同一になるため、実行に必要なパッケージやマニフェストなどの外部ファイルはコントロールノードに集約できます。「コントロールノードとマネージドノードのどちらにあればよいか」の考慮も不要になるため、構成が簡素になります。

Figure 5-4　コントロールノードからクラスタへ直接接続



クラスタへの接続情報は、プレイブックやタスクで指定をしない場合は CLI の kubectl コマンドのデフォルトと同様に、マネージドノードの実行ユーザの$HOME/.kube/config ファイルを参照します。各モジュール共通の kubeconfig パラメータに KUBECONFIG ファイルのパスを指定することで、任意のパスの設定ファイルも使用できます。1 ファイル内に複数のコンテキスト情報がある場合、デフォルトではカレントコンテキストのクラスタへ接続します。コンテキストを指定する場合は、context パラメータにコンテキスト名を指定すれば対象のクラスタへ接続します。

List 5-2　kubeconfig と context を指定したクラスタの指定

```
---
- hosts: localhost
  gather_facts: false

  tasks:
    - name: k8s resource info
      kubernetes.core.k8s_info:
        kind: Pod
        kubeconfig: /path/to/kubeconfig.yaml
        context: my-cluster
```

コントロールノードからクラスタへの接続では、トークンを使った認証も可能です。たとえばクラスタやネームスペースに対する操作の RoleBinding が設定されている ServiceAccount があれば、この ServiceAccount が使用している Secret リソースに含まれる「token」と「ca.crt」を使用します。各モジュール共通の api_key パラメータに「token」の内容をそのまま指定し、ca_cert パラメータには「ca.crt」の内容をファイルへ保存してそのパスを指定します。この 2 つに加えて host パラメータに Kubernetes の API エンドポイントを指定すれば、バインドしてある権限で各モジュールを使用できます。本章で紹介するプレイブックはモジュールに接続情報を指定せず、実行ユーザの情報を使用します。実際の運用では CLI での使用状態に依存あるいは干渉しないように、Ansible 実行用の接続用設定ファイルやトークンを用意した方がよいでしょう。

List 5-3　トークンを使ったクラスタの指定

```
    - name: k8s resource info
      kubernetes.core.k8s_info:
        kind: Pod
        api_key: Secretリソースのtokenの内容
        ca_cert: Secretリソースのca.crtの内容を保存したファイルのパス
        host: https://192.168.0.189:6443
```

5-1-4 コネクションプラグイン

次の節から紹介するプレイブックおよびタスクでは扱いませんが、Docker[*6]と Kubernetes[*7]はそれぞれコネクションプラグインもコレクションで提供されています。

Docker および Kubernetes のコネクションプラグインを使用すると、マネージドノードとして Docker コンテナや Kubernetes の Pod を直接指定できます。マネージドノードへの接続処理をコネクションプラグインで行うため、コンテナ操作用モジュールに限らず「第 3 章 Linux」でも紹介したファイル操作やコマンド実行、ミドルウェア操作など、さまざまなモジュールをコンテナや Pod に対しても使用できます。ただし、Ansible の要件の一つである「マネージドノードに Python インタープリタがインストールされている」という条件があるため、マネージドノードとなるコンテナや Pod に Python インタープリタが必要[*8]です。アプリケーションをコンテナで動作させる場合、コンテナ内は必要最小限のパッケージ構成にするのが基本です。Ansible を使った管理のためだけに Python を追加でインストールする構成は避け、本章で紹介するコンテナ操作用のモジュールを使用することを推奨します。

Docker のコネクションプラグインを使用する場合のインベントリファイルは List 5-4 のとおりです。このインベントリファイルを使用し、プレイブックのターゲットのグループ名に「`containers`」を指定すれば、「`app-container-1`」「`app-container-2`」という名前でデプロイされたコンテナに対して任意のモジュールを実行できます。

List 5-4 docker コネクションプラグインを使用するインベントリファイル

```
1: [containers]
2: app-container-1
3: app-container-2
4:
5: [containers:vars]
6: ansible_connection=community.docker.docker
```

＊6 community.docker.docker – Run tasks in docker containers — Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/community/docker/docker_connection.html`

＊7 kubernetes.core.kubectl – Execute tasks in pods running on Kubernetes. — Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/kubernetes/core/kubectl_connection.html`

＊8 ansible.builtin.raw モジュールのように、Python インタープリタを必要としないモジュールもありますが、この場合は community.docker.docker_container_exec モジュールのようにコンテナ内のコマンド実行用モジュールが利用できます。

5-2 Docker

Docker の CLI コマンドで実行できる「Build/Ship/Run」などの基本的な操作を Ansible で行うためのプレイブックについて紹介します。

5-2-1 docker login を実行する

キーワード

▷ `docker login` コマンド

▷ Docker Hub

▷ community.docker.docker_login モジュール

方 法

Docker Hub へログインするには community.docker.docker_login モジュールを使用します。

List 5-5 community.docker.docker_login 利用例: ./Container/docker_login.yml

```
---
- hosts: docker
  gather_facts: false

  tasks:
    - name: login to Docker Hub
      community.docker.docker_login:
        username: "{{ dockerhub_username }}"
        password: "{{ dockerhub_password }}"
```

解 説

community.docker.docker_login モジュールの主なパラメータは Table 5-4 のとおりです。

Table 5-4 community.docker.docker_login モジュールの主なパラメータ

パラメータ名	説明
username	ユーザ名
password	パスワード
registry_url	レジストリの URL (省略時は Docker Hub)

設定情報は `docker login` 実行時と同じく、デフォルトではマネージドノード上で実行したユーザ

の$HOME/.docker/config.jsonへ保存されます。

ホームディレクトリを使用できないなどの事情で設定ファイルのパスを変更したい場合はconfig_pathパラメータを指定します。

List 5-6 config.jsonファイルの指定例

```
1:    - name: login to Docker Hub
2:      community.docker.docker_login:
3:        username: "{{ dockerhub_username }}"
4:        password: "{{ dockerhub_password }}"
5:        config_path: /path/to/docker/config.json
```

クラウドサービスが提供しているレジストリサービスや、プライベートネットワーク内に独自に運用しているレジストリなど、Docker Hub以外の任意のコンテナレジストリへのログインは、registry_urlパラメータを指定します。

List 5-7 Docker Hub以外のレジストリへのログイン: ./Container/docker_login.yml

```
1:    - name: login to private container registry
2:      community.docker.docker_login:
3:        registry_url: gitlab-ce.example.org:25000
4:        username: "{{ private_registry_username }}"
5:        password: "{{ private_registry_password }}"
```

関 連

▷ community.docker.docker_login – Log into a Docker registry. — Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/community/docker/docker_login_module.html

5-2-2 Dockerイメージをpullする

キーワード

▷ docker pullコマンド

▷ community.docker.docker_imageモジュール

方 法

コンテナイメージをpullするにはcommunity.docker.docker_imageモジュールを使用します。その際、

source パラメータに pull を指定します。

List 5-8　community.docker.docker_image で pull する例: ./Container/docker_pull.yml

```
---
- hosts: docker
  gather_facts: false

  tasks:
    - name: pull image from Docker Hub
      community.docker.docker_image:
        name: centos
        tag: '8'
        source: pull
```

解　説

community.docker.docker_image モジュールでコンテナイメージを pull する場合の主なパラメータは Table 5-5 のとおりです。

Table 5-5　community.docker.docker_image モジュールでコンテナイメージを pull する場合の主なパラメータ

パラメータ名	説明
name	コンテナイメージ名
source	「pull」を指定

source パラメータに pull を指定すると、コンテナイメージをローカルへダウンロードします。pull 済みの場合はタスクの実行ステータスは「ok」となります。

name と tag パラメータは、docker image pull コマンド実行時と同様の書式で、name のみにまとめることもできます。

List 5-9　name パラメータにイメージ名とタグ名を併記

```
    - name: pull image
      community.docker.docker_image:
        name: centos:8
        source: pull
```

Docker Hub 以外のコンテナレジストリから pull する場合は、これも docker image pull コマンド実行時と同様に、リポジトリのアドレスも含めて name パラメータに指定します。

List 5-10　name パラメータにコンテナレジストリのアドレスも記述: ./Container/docker_pull.yml

```
1:    - name: pull image from private container registry
2:      community.docker.docker_image:
3:        name: gitlab-ce.example.org:25000/ansible-user/sample/httpd:latest
4:        source: pull
```

認証が必要なコンテナレジストリの場合は、通常の Docker コマンドと同様にデフォルトではマネージドノード上の実行ユーザの`$HOME/.docker/config.json` の設定が使用されます。

別のパスにある設定ファイルの認証情報を使用したい場合は、環境変数 `DOCKER_CONFIG` にファイルの置かれたディレクトリパスを指定します。認証に使用するファイルは community.docker.docker_login モジュールの config_path パラメータで指定するファイルと同じものになりますが、community.docker.docker_image モジュールの場合はファイルではなく、ディレクトリを指定する点が異なります。環境変数の詳細は Docker のドキュメントを参照してください。

List 5-11　`config.json` ファイルのパスを環境変数で指定

```
1:    - name: pull image from private container registry
2:      community.docker.docker_image:
3:        name: gitlab-ce.example.org:25000/ansible-user/sample/httpd:latest
4:        source: pull
5:      environment:
6:        DOCKER_CONFIG: /path/to/docker/
```

デフォルトの動作では、pull 済みのイメージ更新は行いません。同じタグで更新がある場合は、force_source パラメータに true を指定するとイメージを更新します。force_source パラメータが true の場合でレジストリ上のイメージに更新がない場合はイメージの更新は行われずに、実行ステータスは「ok」となります。

List 5-12　同じタグのイメージの更新

```
1:    - name: pull image
2:      community.docker.docker_image:
3:        name: centos:8
4:        source: pull
5:        force_source: true
```

tag の指定がない場合はデフォルトの latest が使用されます。

関　連

▷ community.docker.docker_image – Manage docker images － Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/community/docker/docker_image_module.html

5-2-3　Docker イメージを build する

キーワード

▷ docker build コマンド
▷ Dockerfile
▷ community.docker.docker_image モジュール

方　法

コンテナイメージを build するには community.docker.docker_image モジュールを使用し、source パラメータに build を指定します。

docker image build 同様に build 用の Dockerfile を別途用意し、build のサブパラメータの path に Dockerfile のあるディレクトリパスを指定します。

List 5-13　community.docker.docker_image で build する例: ./Container/docker_build.yml

```
---
- hosts: docker
  gather_facts: false

  vars:
    local_path: docker_src/
    remote_path: /var/tmp/docker/build

  tasks:
    - name: create directory
      ansible.builtin.file:
        state: directory
        path: "{{ remote_path }}"

    - name: copy source
      ansible.builtin.copy:
        src: "{{ local_path }}"
```

```
18:         dest: "{{ remote_path }}"
19:
20:     - name: build image
21:       community.docker.docker_image:
22:         name: nginx_sample:0.1
23:         source: build
24:         build:
25:           path: "{{ remote_path }}"
```

解　説

community.docker.docker_image モジュールでコンテナイメージを build する場合の主なパラメータは Table 5-6 のとおりです。

Table 5-6　community.docker.docker_image モジュールでコンテナイメージを pull する場合の主なパラメータ

パラメータ名		説明
name		build するイメージ名
source		「build」を指定
build		イメージビルド時のパラメータ指定
	path	Dockerfile のあるパス

Dockerfile やイメージへコピーするファイルなどのイメージビルドに使用する資材は、このタスクを実行するマネージドノード上にあらかじめ配置しておく必要があります。

build に必要な資材は別途転送したり、Git リポジトリから clone するタスクなどを使って準備し、build のサブパラメータの path でパスを指定します。

build するイメージ名は name に指定します。pull の場合と同様に name と tag を分けて指定することもできます。

List 5-14　イメージ名とタグ名を name パラメータと tag パラメータにそれぞれ記述:
./Container/docker_build.yml

```
1:     - name: build image
2:       community.docker.docker_image:
3:         name: nginx_sample
4:         tag: "0.1"
5:         source: build
6:         build:
7:           path: "{{ remote_path }}"
```

build に使用する Dockerfile の書式などの内容については、Docker の公式サイトやマニュアルを参照してください。

関　連

▷ Dockerfile reference | Docker Documentation

https://docs.docker.com/engine/reference/builder/

▷ community.docker.docker_image – Manage docker images — Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/community/docker/docker_image_module.html

5-2-4　Docker イメージを tar アーカイブへ save する

キーワード

▷ docker image save コマンド

▷ community.docker.docker_image モジュール

方　法

コンテナイメージを tar アーカイブへ save するには、community.docker.docker_image モジュールを使用し、archive_path パラメータに出力先 tar アーカイブのパスを指定します。

List 5-15　community.docker.docker_image でイメージファイルを tar アーカイブ出力する例: ./Container/docker_save_and_load.yml

```
---
- hosts: docker
  gather_facts: false

  vars:
    image_remote_path: /var/tmp/image/sample_image.tar
    image_local_path: /var/tmp/image/sample_image.tar

  tasks:
    - name: create directory for iamge file
      ansible.builtin.file:
        path: "{{ image_remote_path | ansible.builtin.dirname }}"
        state: directory

```

```
15:     - name: save image to file
16:       community.docker.docker_image:
17:         name: nginx_sample
18:         tag: "0.1"
19:         source: local
20:         archive_path: "{{ image_remote_path }}"
```

解　説

community.docker.docker_image モジュールでコンテナイメージを save する場合の主なパラメータは Table 5-7 のとおりです。

Table 5-7　community.docker.docker_image モジュールでコンテナイメージを save する場合の主なパラメータ

パラメータ名	説明
name	tar 出力対象イメージ名
source	対象イメージの取得場所（解説参照）
archive_path	イメージの出力先パス

コンテナイメージを tar アーカイブへ出力するには archive_path パラメータを使用します。

pull 済みあるいは build 済みのコンテナイメージを tar アーカイブへ保存する場合は、前掲のプレイブックのように source パラメータに local を指定することでイメージファイルへ出力ができます。

コンテナイメージの pull と tar アーカイブへの save を 1 つのタスクにまとめることもできます。source パラメータに pull を指定し、archive_path に出力先の tar アーカイブのパスを指定することで、コンテナイメージの pull と save を 1 つのタスクで行います。

List 5-16　pull と save をまとめて行う例: ./Container/docker_pull_and_save.yml

```
1:  vars:
2:    image_file_path: /var/tmp/image/centos-8.tar
3:
4:  tasks:
5:    - name: create directory for image file
6:      ansible.builtin.file:
7:        path: "{{ image_file_path | ansible.builtin.dirname }}"
8:        state: directory
9:
10:   - name: pull image and save
```

```
11:       community.docker.docker_image:
12:         name: "centos"
13:         tag: "8"
14:         source: pull
15:         archive_path: "{{ image_file_path }}"
```

コンテナイメージの build と tar アーカイブへの save もまとめることができます。source パラメータに build を指定し、archive_path パラメータに出力先を指定することで、build 完了後に tar アーカイブの作成を 1 タスクで行います。

List 5-17　build と save をまとめて行う例: ./Container/docker_build_and_save.yml

```
1:     - name: build image and save
2:       community.docker.docker_image:
3:         name: nginx_sample:0.1
4:         source: build
5:         build:
6:           path: "{{ remote_path }}"
7:         archive_path: "{{ image_file_path }}"
```

関　連

▷ community.docker.docker_image – Manage docker images － Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/community/docker/docker_image_module.html

5-2-5　tar アーカイブの Docker イメージファイルを load する

キーワード

▷ docker image save コマンド
▷ docker image load コマンド

方　法

tar アーカイブで保存されたコンテナイメージを load するには community.docker.docker_image モジュールを使用し、source パラメータに load を指定します。

List 5-18　community.docker.docker_image で tar アーカイブのイメージファイルを load する例: ./Container/docker_save_and_load.yml

```
---
- hosts: docker
  gather_facts: false

  tasks:
    - name: load image from file
      community.docker.docker_image:
        name: nginx_sample
        tag: "0.1"
        load_path: "{{ image_local_path }}"
        source: load
```

解　説

community.docker.docker_image モジュールでコンテナイメージを load する場合の主なパラメータは Table 5-8 のとおりです。

Table 5-8　community.docker.docker_image モジュールでコンテナイメージを load する場合の主なパラメータ

パラメータ名	説明
name	コンテナイメージ名
source	「load」を指定
load_path	イメージファイルのパス

tar アーカイブのイメージファイルは、Docker コマンドの場合は docker image load コマンドを使用してローカルストレージへ展開します。Ansible では community.docker.docker_image モジュールを使用して同様の操作ができます。

community.docker.docker_image モジュールでコンテナイメージを load するには source パラメータに load を指定し、load_path パラメータにイメージファイルの tar アーカイブのパスを指定します。load する tar アーカイブはマネージドノードのファイルシステム上にある必要があるため、コントロールノードからコピーなどで配置しておきます。

関　連

▷ community.docker.docker_image – Manage docker images － Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/community/docker/docker_image_module.html

5-2-6　Docker イメージを push する

キーワード

▷ `docker image push` コマンド

▷ community.docker.docker_image モジュール

方　法

pull 済みや build 済みのコンテナイメージをリポジトリへ push するには、community.docker.docker_image モジュールを使用し、push パラメータに true を指定します。

`docker image ls` コマンドで確認できるローカルストレージに保持されているコンテナイメージを push するには List 5-19 のように source パラメータに local を指定します。

List 5-19　community.docker.docker_image で push する例: ./Container/docker_push.yml

```
---
- hosts: docker
  gather_facts: false

  tasks:
    - name: push image to Docker Hub
      community.docker.docker_image:
        name: nginx_sample
        tag: "0.1"
        repository: "{{ dockerhub_repository_path }}"
        push: true
        source: local
```

解　説

community.docker.docker_image モジュールでコンテナイメージを push する場合の主なパラメータは Table 5-9 のとおりです。

Table 5-9　community.docker.docker_image モジュールでコンテナイメージを push する場合の主なパラメータ

パラメータ名	説明
name	push 対象のイメージ名
source	対象イメージの取得場所（解説参照）
push	true を指定
repository	push 先のリポジトリ URL

push パラメータを true にすることで、name に指定したイメージを repository パラメータに指定したコンテナリポジトリへ push します。

Docker Hub ではなく、任意のコンテナレジストリに push する場合は、pull の場合と同様に repository パラメータの指定にコンテナレジストリのアドレスも含めて指定します。

List 5-20　コンテナレジストリのアドレスを指定して push する例: ./Container/docker_push.yml

```
1:    - name: push image to private container registry
2:      community.docker.docker_image:
3:        name: nginx_sample:0.1
4:        repository: gitlab-ce.example.org:25000/ansible-user/sample/nginx_sample:0.1
5:        push: true
6:        source: local
```

source パラメータに build を指定することで、Docker イメージを build した結果を push できます。イメージの build と push という 2 つの作業を 1 つのタスクにまとめることができます。

List 5-21　build と push をまとめて行う例: ./Container/docker_build_and_push.yml

```
1:    - name: build image and push
2:      community.docker.docker_image:
3:        name: nginx_sample:0.1
4:        source: build
5:        build:
6:          path: "{{ remote_path }}"
7:        repository: gitlab-ce.example.org:25000/ansible-user/sample/nginx_sample:0.1
8:        push: true
```

同様に、source パラメータに load を指定することで、tar アーカイブ化された Docker イメージファイルの load と push をまとめることもできます。List 5-22 のタスクは、まずコントロールノード上にあるイメージファイルをマネージドノードへコピーし、次にイメージの load と push を 1 つのタスクで実行します。

List 5-22　load と push をまとめて行う例: ./Container/docker_load_and_push.yml

```
1:    - name: copy image file
2:      ansible.builtin.copy:
3:        src: "{{ lookup('ansible.builtin.env', 'HOME') }}/alpine-3.12.tar"
4:        dest: "{{ work_dir }}"
5:
6:    - name: load image and push
```

```
 7:      community.docker.docker_image:
 8:        name: alpine:3.12
 9:        repository: gitlab-ce.example.org:25000/ansible-user/sample/alpine:3.12
10:        push: true
11:        load_path: "{{ work_dir }}/alpine-3.12.tar"
12:        source: load
```

関　連

▷ 5-2-3 Docker イメージを build する

▷ 5-2-5 tar アーカイブの Docker イメージファイルを load する

▷ community.docker.docker_image - Manage docker images － Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/community/docker/docker_image_module.html

5-2-7　ブリッジネットワークを作成する

キーワード

▷ docker network コマンド

▷ community.docker.docker_network モジュール

方　法

Docker ネットワークを作成するには、community.docker.docker_network モジュールを使用します。本項ではブリッジネットワークの作成について説明します。

List 5-23　community.docker.docker_network モジュールの利用例: ./Container/docker_run.yml

```
1: ---
2: - hosts: docker
3:   gather_facts: false
4:
5:   tasks:
6:     - name: create docker network
7:       community.docker.docker_network:
8:         name: my_docker_network
9:         driver: bridge
```

解　説

Docker ではコンテナ間で相互に通信を行う場合などは、ユーザ定義ブリッジネットワークを作成します。このネットワークは Docker の内部 DNS が有効になるため、コンテナ名を使ってコンテナ同士が相互通信できるようになります。

多くの場合に使用されるブリッジネットワークを作成するには、Table 5-10 のパラメータを使用します。

Table 5-10　community.docker.docker_network モジュールでブリッジネットワーク作成する場合の主なパラメータ

パラメータ名	説明
name	ネットワーク名
driver	bridge を指定

応　用

作成済みコンテナに対してブリッジネットワークを追加することもできます。作成済みコンテナ名またはコンテナ ID を connected パラメータにリスト指定すると、実行中コンテナをネットワークに追加できます。

connected を指定したコンテナを追加するデフォルトの動作は、マージではなく、指定したコンテナのみが接続された状態にします。appends パラメータに true を指定するとマージします。次の例では appends が有効なため、my_nginx_network ネットワークに指定した 2 つのコンテナが追加で接続されます。appends が無効の場合は、すでに my_nginx_network ネットワークに接続していたほかのコンテナがある場合は、それらのコンテナは切断されるため注意してください。

List 5-24　Docker ネットワークにコンテナを接続する

```
1:    - name: create docker network and append container
2:      community.docker.docker_network:
3:        name: my_nginx_network
4:        driver: bridge
5:        connected:
6:          - my_nginx_container1
7:          - my_nginx_container2
8:        appends: true
```

作成する Docker ネットワークのサブネットアドレスなどを指定する場合は、ipam_config パラメータで指定を行います。通常は Docker が自動で割り当てるため、明示的に指定する必要はありません

が、デプロイするコンテナのゲートウェイの指定や IP アドレスを固定したい場合などに使用します。詳細は本モジュールのドキュメントの ipam_config の項を参照してください。

List 5-25　サブネットアドレスを指定した Docker ネットワークの作成: ./Container/docker_run.yml

```
    - name: create docker network
      community.docker.docker_network:
        name: my_docker_network
        driver: bridge
        ipam_config:
          - subnet: 10.21.0.0/24
```

関　連

▷ community.docker.docker_network – Manage Docker networks — Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/community/docker/docker_network_module.html

5-2-8　コンテナをデプロイする

キーワード

▷ docker container run コマンド

▷ community.docker.docker_container モジュール

方　法

コンテナをデプロイするには community.docker.docker_container モジュールを使用します。

List 5-26　community.docker.docker_container モジュールの利用例: ./Container/docker_run.yml

```
---
- hosts: docker
  gather_facts: false

  tasks:
    - name: run postgres container
      community.docker.docker_container:
        name: my_postgres
        image: postgres:13-alpine
        env:
```

```
11:           POSTGRES_DB: sample_db
12:           POSTGRES_USER: user
13:           POSTGRES_PASSWORD: secret
14:         volumes:
15:           - pgdata:/var/lib/postgresql/data
```

解　説

community.docker.docker_container の主なパラメータは Table 5-11 のとおりです。

Table 5-11　コンテナをデプロイするときの主なパラメータ

パラメータ名	説明
name	コンテナ名
image	イメージ名
networks	ネットワーク設定
volumes	ボリューム設定
volumes_from	ボリューム共有設定
published_ports	ポートの公開設定
restart_policy	再起動ポリシーの設定
env	環境変数設定
command	コンテナ起動時に実行するコマンド

Docker ネットワークを指定する場合は networks パラメータの name サブパラメータに作成済みネットワーク名を指定します。ここで指定するネットワーク名は、あらかじめ作成しておかないとエラーになるため注意してください。

List 5-27　Docker ネットワークを指定したコンテナのデプロイ: ./Container/docker_run.yml

```
1:     - name: run nginx container
2:       community.docker.docker_container:
3:         name: my_nginx
4:         image: nginx_sample:0.1
5:         networks:
6:           - name: my_docker_network
```

コンテナから使用するボリュームは volumes パラメータにリストで設定します。ボリュームの設定はほかに mount パラメータもありますが本項では volumes パラメータについて説明します。Docker の CLI コマンドの-v または--volume に相当します。Docker のボリュームをマウントする場合は、「ボ

リューム名:マウントポイント」と指定します。このときボリュームが事前に存在していない場合は、コンテナ作成時に自動で作成されます。指定したボリュームが存在する場合は指定したボリュームがコンテナでマウントされ、そこに保存してあるファイルを参照できます。

ホスト OS のファイルシステムを共有するバインドマウントを使用する場合は、「ホスト OS のパス:マウントポイント」と指定します。デプロイされたコンテナはホスト OS の指定パスを参照できるようになります。ホスト OS に指定のパスが存在しなかった場合は自動で作成されます。

List 5-28　ボリュームを指定したコンテナのデプロイ: ./Container/docker_run.yml

```
1:    - name: run nginx container
2:      community.docker.docker_container:
3:        name: my_nginx
4:        image: nginx_sample:0.1
5:        volumes:
6:          - log-data:/var/log/nginx/
7:          - /opt/www:/usr/share/nginx/html/hostdata
```

リモートからの接続を処理する HTTP サーバなどのコンテナにおいて、ホスト OS 以外のリモートホストにもサービスを提供したい場合は、published_ports パラメータを使用します。Docker CLI コマンドの-p または--publish に相当します。このパラメータはホスト OS でリッスンするポートと、リッスンしたポートへのアクセスをコンテナのどのポートへ転送するかを指定することで、リモートホストからホスト OS を経由してコンテナのサービスへアクセスできるようになります。書式は「ホスト OS でリッスンするポート:転送先のコンテナのポート」と指定します。

List 5-29　ホストに公開するポートを指定したコンテナのデプロイ: ./Container/docker_run.yml

```
1:    - name: run nginx container
2:      community.docker.docker_container:
3:        name: my_nginx
4:        image: nginx_sample:0.1
5:        published_ports:
6:          - 8080:80
```

restart_policy パラメータはコンテナのクラッシュや Docker エンジンを再起動した際に、自動でコンテナを起動するかどうかの設定です。デフォルトは no が設定されており、自動で再起動されません。Docker エンジンの再起動や、ホスト OS の再起動後の Docker 起動時に自動で起動させたい場合は always を指定します。詳細は Docker のドキュメントを参照してください。

env はデプロイするコンテナ内の環境変数を設定します。環境変数名とその値を指定します。

command はコンテナで実行するコマンドを上書きできます。Docker の CLI コマンドで、イメージ名の後に指定できるコマンド名と同様に機能します。command パラメータにコマンドとオプションをすべて文字列で指定することも、コマンドと引数をそれぞれ 1 つずつリストにして指定することもできます。

List 5-30　実行するコマンドを指定したコンテナのデプロイ: ./Container/docker_run.yml

```
    - name: run centos container
      community.docker.docker_container:
        name: my_centos
        image: centos:7
        command:
          - tail
          - -f
          - /dev/null
```

応　用

community.docker.docker_container モジュール利用の応用として、ボリュームをバックアップするコンテナデプロイの例について紹介します。対象コンテナを停止し、ファイルシステムレベルでファイルバックアップができます。state パラメータに stopped を指定しコンテナを停止します。volumes_from パラメータを使用するとパラメータで指定したコンテナのボリュームをマウントできます。データをバックアップしたいコンテナの state パラメータを stopped に指定し、ワーク用のコンテナを別途用意し volumes_from を使用すると、停止しているコンテナのボリュームをワーク用コンテナで `tar` コマンドを実行してバックアップできます。

volumes_froms パラメータは `docker run` コマンドの`--volumes-from` の機能を提供します。詳細は Docker のドキュメントを参照してください。

List 5-31　コンテナボリュームを共有して tar バックアップする例: ./Container/docker_run_volumefrom.yml

```
- name: stop postgres container
  community.docker.docker_container:
    name: my_postgres
    state: stopped

- name: run centos container
  community.docker.docker_container:
    name: work_container
    image: centos:7
    command: tar cvf /backup/backup.tar /var/lib/postgresql/data
```

```
    volumes_from:
      - my_postgres
    volumes:
      - /tmp/container/postgres:/backup
    state: started
    auto_remove: true

- name: start postgres container
  community.docker.docker_container:
    name: my_postgres
    state: started
```

本タスクはコンテナ内のファイルをバックアップする例です。使用しているアプリケーションやミドルウェアでバックアップ用のツールが用意されている場合は、それらを使用できないかをまず検討してみてください。また、state パラメータに absent を指定するとコンテナは削除されます。

関　連

▷ community.docker.docker_container – manage docker containers — Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/community/docker/docker_container_module.html`

5-2-9　コンテナ内のコマンドを実行する

キーワード

▷ `docker container exec` コマンド
▷ community.docker.docker_container_exec モジュール

方　法

実行中コンテナ内の任意のコマンドを実行するには community.docker.docker_container_exec モジュールを使用します。

List 5-32　community.docker.docker_container_exec モジュールの利用例: ./Container/docker_exec.yml

```
---
- hosts: docker
  gather_facts: false
```

```
4:
5:  tasks:
6:    - name: run command
7:      community.docker.docker_container_exec:
8:        container: work_container
9:        command: sh -c "cat - > sample.txt"
10:       chdir: /var/tmp
11:       stdin: |
12:         Hello
13:         Ansible
14:       stdin_add_newline: false
```

解　説

community.docker.docker_container_exec モジュールを使うと、docker container exec を使用したコマンド実行と同等の処理ができます。

主なパラメータは Table 5-12 のとおりです。

Table 5-12　コンテナ内のコマンドを実行するときの主なパラメータ

パラメータ名	説明
container	実行対象のコンテナ名
command	実行するコマンド
chdir	実行時のディレクトリ
stdin	コマンド実行時の標準入力
stdin_add_newline	stdin 指定時に末尾の改行を追加するかどうか

実行するコマンドは command パラメータで指定します。コマンド実行時のワーキングディレクトリは chdir パラメータで指定できます。

stdin パラメータに文字列を指定すると、コマンド実行の標準入力となります。このとき stdin で指定した内容の末尾に自動で改行が付与されます。この改行の自動付与を抑止したい場合は stdin_add_newline パラメータに false を指定します。stdin_add_newline パラメータはデフォルトで true です。

補　足

ansible.builtin.shell モジュールのようにシェルは使っていないため、「ls > /tmp/ls.txt」のようにリダイレクトなどは使用できません。もしファイルへのリダイレクトを行いたい場合はコンテナ内の sh などのシェルコマンドを使って、「sh -c "echo 'abc' > /tmp/abc"」のように指定します。

関　連

▷ community.docker.docker_container_exec – Execute command in a docker container － Ansible Documentation
https://docs.ansible.com/ansible/latest/collections/community/docker/docker_container_exec_module.html

5-2-10　Docker Compose を使ってコンテナをデプロイする

キーワード

▷ Docker Compose
▷ community.docker.docker_compose モジュール

方　法

community.docker.docker_compose モジュールで Docker Compose を使ったコンテナデプロイができます。

List 5-33　community.docker.docker_compose モジュールの利用例: ./Container/docker_compose.yml

```
---
- hosts: docker
  gather_facts: false

  tasks:
    - name: exec compose inline
      community.docker.docker_compose:
        project_name: sample
        definition:
          version: '3'
          services:
            my_httpd:
              image: httpd:latest
              ports:
                - 8888:80
            my_centos:
              image: centos:7
              command:
                - "tail"
                - "-f"
                - "/dev/null"
```

解　説

このモジュールは、別途作成した Compose File を指定する方法と、モジュールのパラメータとして Compose File の内容相当をプレイブック内に記載する方法があります。

主なパラメータは Table 5-13 のとおりです。

Table 5-13　community.docker.docker_compose モジュールの主なパラメータ

パラメータ名	説明
definition	Compose の定義を記述する
project_name	コンテナ名などのベースになるプロジェクト名
project_src	Compose File のパス

モジュールのパラメータとして Compose File の内容をタスク内に記述する場合は、definition パラメータを使用します。definition を指定する場合は、project_name の指定も必須になり、デプロイ時のプロジェクト名になります。これは Compose File を使った従来のデプロイの場合の、デフォルトでディレクトリ名が使用される項目です。

一方ですでに Compose File の資産があり、それを Ansible を使って管理したい場合は、既存の Compose File をそのまま再利用できます。project_src パラメータに Compose File のパスを指定します。

Compose File は実行するマネージドノード上に存在する必要があるため、プレイブックと Compose File を同じリポジトリで管理している場合などは事前に Compose File をマネージドノードへ転送する必要があることに注意してください。

project_src で指定するパスに、`docker-compose.yml` ファイルを配置してください。project_src を使う場合は、コンテナ名のベースとなるプロジェクト名は project_src で指定したディレクトリ名です。`docker-compose -p` に相当するプロジェクト名の指定を行う場合は、project_src 使用時は任意ですが project_name パラメータで指定できます。

List 5-34　project_src で既存の Compose File を指定する例: ./Container/docker_compose.yml

```
1: vars:
2:   compose_file_localpath: compose_sample/
3:   compose_file_remotepath: /var/tmp/compose_sample
4:
5: tasks:
6:   - name: copy source
7:     ansible.builtin.copy:
8:       src: "{{ compose_file_localpath }}"
```

```
 9:      dest: "{{ compose_file_remotepath }}"
10:
11:  - name: exec compose
12:    community.docker.docker_compose:
13:      project_src: "{{ compose_file_remotepath }}"
14:      state: present
```

応　用

GitHub リポジトリで公開されている Compose File を使ったコンテナデプロイであれば、ansible.builtin.git モジュールで Git クローンを行い Compose File があるディレクトリを指定することで、コンテナを使ってデプロイできるように公開されているさまざまな OSS アプリケーションを Ansible を使って管理することもできます。

クローズドな環境であれば、プライベート Git リポジトリから必要な資材をクローンして同様のことができます。

例として OSS の IPAM ツールである NetBox の Docker Compose 版[*9]をデプロイするには、List 5-35 のようなプレイブックになります。

List 5-35　NetBox のデプロイ : ./Container/docker_compose.yml

```
 1:    - name: clone git repository
 2:      ansible.builtin.git:
 3:        repo: https://github.com/netbox-community/netbox-docker.git
 4:        dest: ~/local/netbox-docker
 5:        version: release
 6:
 7:    - name: create override file
 8:      ansible.builtin.copy:
 9:        content: |
10:          version: '3.4'
11:          services:
12:            netbox:
13:              ports:
14:                - 8000:8080
15:        dest: ~/local/netbox-docker/docker-compose.override.yml
16:
17:    - name: exec compose
```

＊9　netbox-community/netbox
https://github.com/netbox-community/netbox （netbox-docker 1.2.0、NetBox v2.11.11）

```
18:     community.docker.docker_compose:
19:       project_src: ~/local/netbox-docker
20:       state: present
```

Docker Compose で構成されたコンテナを削除するには state パラメータに absent を指定します。削除ではなく停止状態にするには、stopped パラメータに true を指定します。community.docker.docker_container モジュールとは異なり、state パラメータの取りうる値は present と absent の 2 つの値です。

補　足

実行にはマネージドノードに Python の docker-compose パッケージが必要です。

`pip install docker-compose` でインストールする必要があります。

関　連

▷ Overview of Docker Compose | Docker Documentation

`https://docs.docker.com/compose/`

▷ community.docker.docker_compose – Manage multi-container Docker applications with Docker Compose. — Ansible Documentation

`https://docs.ansible.com/ansible/latest/collections/community/docker/docker_compose_module.html`

▷ netbox-community/netbox

`https://github.com/netbox-community/netbox`

5-3　Kubernetes

本節では、Kubernetes クラスタに対するリソース作成や情報取得などを行うプレイブックを紹介します。マニフェストファイルのクラスタへの適用はもちろん、`kubectl` コマンドの `get` や `logs` を使った情報取得も Ansible で自動化できます。

5-3-1　Kubernetes のリソースを作成する

キーワード

▷ `kubectl apply` コマンド

▷ Jinja2

▷ kubernetes.core.k8s モジュール

方　法

Kubernetes クラスタへリソースの作成を行うには、kubernetes.core.k8s モジュールを使用します。kubeclt コマンドの create や apply を使ったリソース作成に相当します。

マニフェストファイルを指定してリソース作成するプレイブックは List 5-36 のとおりです。

List 5-36　マニフェストファイルを指定する例: ./Container/k8s_apply_resource.yml

```
---
- hosts: k8s
  gather_facts: false

  vars:
    target_namespace: example
    manifest_file_localpath: kubernetes_src/deployment.yaml
    manifest_file_remotepath: /var/tmp/kubernetes/deployment.yaml

  tasks:
    - name: copy manifest file
      ansible.builtin.copy:
        src: "{{ manifest_file_localpath }}"
        dest: "{{ manifest_file_remotepath }}"

    - name: create deployment from a local file
      kubernetes.core.k8s:
        namespace: "{{ target_namespace }}"
        src: "{{ manifest_file_remotepath }}"
        state: present
```

kubernetes.core.k8s モジュールを使用したリソースの作成はいくつかの方法があるので、それぞれ解説します。

解　説

kubernetes.core.k8s モジュールの主なパラメータは Table 5-14 のとおりです。リソース作成の方法によって、それぞれ使用するパラメータが異なります。

Table 5-14　kubernetes.core.k8s モジュールの主なパラメータ

パラメータ名	説明
namespace	デプロイ先ネームスペース
name	作成するリソース名
api_version	リソースの API バージョン
kind	リソース種別
src	デプロイするマニフェストファイルのパス
template	デプロイするリソース定義のテンプレートファイルのパス
resource_definition	マニフェストの定義のインライン定義

リソース作成の方法は以下のとおりです。

- kubernetes.core.k8s モジュールのパラメータ指定
- マニフェストファイル指定
- kubernetes.core.k8s モジュールの resource_definition パラメータ指定
- テンプレートを使ったマニフェスト指定

◇ kubernetes.core.k8s モジュールのパラメータ指定

kubernetes.core.k8s モジュールのパラメータ指定では、name に作成するリソース名、kind にリソースの種類、そして api_version に API バージョンを指定します。追加情報のない Namespace リソースのように、リソース名のみでよいリソースであればこの方式で作成できます。

List 5-37　kubernetes.core.k8s モジュールのパラメータでリソース作成する例:
./Container/k8s_apply_resource.yml

```
- name: create namespace
  kubernetes.core.k8s:
    name: "{{ target_namespace }}"
    kind: Namespace
    api_version: v1
    state: present
```

◇ マニフェストファイル指定

冒頭の例のように `kubectl apply -f` などで使用するマニフェストファイルがすでにある場合は、src にマニフェストファイルのパスを指定することでリソースを作成できます。マニフェストファイルはマネージドノードのファイルシステム上に配置されている必要があります。`kubectl apply` コマン

ドとは異なり、ディレクトリや URL を指定するとエラーになるため、複数のマニフェストファイルがある場合は個別に指定、もしくは事前にダウンロードする必要があります。

マニフェストファイルの管理をプレイブックとは別の専用のリポジトリで行っている構成であれば、Git クローンするタスクを作成してから、この方法でリソース作成するとよいでしょう。コントロールノードにプレイブックと一緒にマニフェスト管理も保持しているのであれば、ansible.builtin.copy モジュールで転送します。

List 5-38　マネージドノード上のファイルを src パラメータで指定する: ./Container/k8s_apply_resource.yml

```
- name: create directory for manifest
  ansible.builtin.file:
    state: directory
    path: "{{ manifest_file_remotepath | ansible.builtin.dirname }}"

- name: copy manifest file
  ansible.builtin.copy:
    src: "{{ manifest_file_localpath }}"
    dest: "{{ manifest_file_remotepath }}"

- name: create deployment from a local file
  kubernetes.core.k8s:
    namespace: "{{ target_namespace }}"
    src: "{{ manifest_file_remotepath }}"
    state: present
```

◇ kubernetes.core.k8s モジュールの resource_definition パラメータ指定

マニフェストファイルのパスを指定するのではなく、resource_definition パラメータを使用してマニフェストファイルの内容をモジュールのパラメータにインラインで記述もできます。ただし resource_definition パラメータへのインライン記述は、定義内に YAML の開始を表す「---」を使用できないため、マニフェストファイルでは可能だった 1 ファイル内の複数定義をインラインで表現できません。複数の定義がある場合は、定義ごとにタスクを作成します。

List 5-39　resource_definition パラメータでインラインでリソース定義する:
./Container/k8s_apply_resource.yml

```
- name: create service from an inline definition
  kubernetes.core.k8s:
    state: present
    resource_definition:
```

```
 5:      apiVersion: v1
 6:      kind: Service
 7:      metadata:
 8:        labels:
 9:          app: sample-http
10:        name: sample-http
11:        namespace: "{{ target_namespace }}"
12:      spec:
13:        ports:
14:          - port: 80
15:            protocol: TCP
16:            targetPort: 80
17:            name: http
18:        selector:
19:          app: sample-http
20:        type: NodePort
```

◇ テンプレートを使ったマニフェスト指定

マニフェストファイルそのものではなく、template パラメータを使うと Jinja2 テンプレートの書式を使ったファイルでもリソースを作成できます。

前述のマニフェストファイルの指定の場合は、従来どおりの静的なファイルを指定しますが、テンプレートを使えば、可変にしたい箇所を変数参照するように記述したファイルを利用できます。そのため、1 つのテンプレートファイルで変数を使い分けて、環境などに応じて Kubernetes リソースを作成できます。

src パラメータを使ったマニフェストファイルの指定は、マネージドノード上のパスを指定します。一方、template パラメータに指定するテンプレートファイルは、コントロールノードのファイルです。ansible.builtin.template モジュール同様に、通常は `templates` ディレクトリ以下にファイルを置くとよいでしょう。

List 5-40　template パラメータを使ったテンプレートファイルの指定: ./Container/k8s_apply_resource.yml

```
1: - name: create resource by template file from controll node file system
2:   kubernetes.core.k8s:
3:     state: present
4:     template: kubernetes_template/deployment.yaml.j2
5:   vars:
6:     k8s_template_namespace: template-ns
7:     k8s_template_resourcename: template-http
8:     k8s_template_replica: 2
```

上記の例で指定したテンプレートファイルは List 5-41 のとおりです。変数参照するように記述した箇所は、タスクの vars で定義した変数の値が使用されます。

List 5-41　Jinaj2 を使ったマニフェストのテンプレート例:
./Container/templates/kubernetes_template/deployment.yaml.j2

```
---
apiVersion: v1
kind: Namespace
metadata:
  name: {{ k8s_template_namespace }}
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: template-sample
  name: {{ k8s_template_resourcename }}
  namespace: {{ k8s_template_namespace }}
spec:
  replicas: {{ k8s_template_replica }}
  selector:
    matchLabels:
      app: template-sample
  template:
    metadata:
      labels:
        app: template-sample
    spec:
      containers:
      - image: httpd
        name: httpd
```

応　用

GitHub で公開されているマニフェストからリソースを作成する例として、任意の Kubernetes クラスタで LoadBalancer タイプの Service が使えるようになる MetalLB[*10]をデプロイするプレイブックは List 5-42 のとおりです。GitHub 上の 2 つのマニフェストのファイルを ansible.builtin.get_url モジュールを使用してダウンロードしてリソース作成し、MetalLB で使用するアドレス設定の ConfigMap リソースを作成します。使用するアドレス設定は環境に合わせてください。

＊ 10　MetalLB, bare metal load-balancer for Kubernetes
`https://metallb.universe.tf/`（MetalLB v0.10.2）

List 5-42　MetalLB のデプロイ: ./Container/k8s_apply_resource.yml

```
 1:     - name: download MetalLB manifest
 2:       ansible.builtin.get_url:
 3:         url: "{{ item }}"
 4:         dest: /var/tmp/kubernetes
 5:         mode: '0664'
 6:       loop:
 7:         - https://raw.githubusercontent.com/metallb/metallb/v0.10.2/manifests/⇒
 8: namespace.yaml
 9:         - https://raw.githubusercontent.com/metallb/metallb/v0.10.2/manifests/⇒
10: metallb.yaml
11:     - name: deploy MetalLB
12:       kubernetes.core.k8s:
13:         src: "{{ item }}"
14:         state: present
15:       loop:
16:         - /var/tmp/kubernetes/namespace.yaml
17:         - /var/tmp/kubernetes/metallb.yaml
18:     - name: create config
19:       kubernetes.core.k8s:
20:         state: present
21:         resource_definition:
22:           apiVersion: v1
23:           kind: ConfigMap
24:           metadata:
25:             name: config
26:             namespace: metallb-system
27:           data:
28:             config: |
29:               address-pools:
30:                 - name: default
31:                   protocol: layer2
32:                   addresses:
33:                     - 172.21.240.21-172.21.240.120
```

また、state パラメータに absent を指定すると、指定した Kubernetes リソースを削除できます。`kubectl delete` 相当の操作なので、Pod の削除などは Deployment リソースの定義などによってはセルフヒーリングが行われます。

補　足

src、template、resource_definition パラメータについては、1 つのタスクで使用できるのはどれか 1 つです。併用はできません。

関　連

▷ kubernetes.core.k8s – Manage Kubernetes (K8s) objects － Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/kubernetes/core/k8s_module.html

▷ MetalLB, bare metal load-balancer for Kubernetes

https://metallb.universe.tf/

5-3-2　Pod 情報や一覧を取得する

キーワード

▷ kubectl get コマンド

▷ kubernetes.core.k8s_info モジュール

方　法

kubectl get に相当する Kubernetes リソースの情報を取得するには kubernetes.core.k8s_info モジュールを使います。

List 5-43　kubernetes.core.k8s_info モジュールの例: ./Container/k8s_info.yml

```
---
- hosts: k8s
  gather_facts: false

  tasks:
    - name: get resource info
      kubernetes.core.k8s_info:
        namespace: monitoring
        kind: Pod
      register: k8s_info_result
```

解　説

kubernetes.core.k8s_info モジュールの主なパラメータは Table 5-15 のとおりです。

Table 5-15　kubernetes.core.k8s_info モジュールの主なパラメータ

パラメータ名	説明
kind	対象リソースタイプ
namespace	対象ネームスペース

name	対象リソース名

・kind パラメータに、Pod や Service など情報取得したい Kubernetes リソースタイプ名を指定します。
・namespace パラメータには対象ネームスペース名を指定します。省略した場合*11はクラスタ全体を対象に情報収集します。
・name パラメータにはリソース名を指定します。省略した場合は対象ネームスペースの全リソースの情報を収集します。

monitoring ネームスペースで実行中の Pod 情報を取得するには List 5-44 のように記述します。取得される情報は構造化データとなっているため、各種フィルタなどを使って必要な情報の抜き出しや加工ができます。Pod 情報一覧から Pod 名のみを出力するには map フィルタを使用できます。

List 5-44　monitoring ネームスペースの Pod 情報を取得する例: ./Container/k8s_info.yml

```
- name: get resource info
  kubernetes.core.k8s_info:
    namespace: monitoring
    kind: Pod
  register: k8s_info_result

- name: print result
  debug:
    msg: '{{ k8s_info_result.resources | map(attribute="metadata.name") }}'
```

Kubernetes リソースに設定されているラベルを label_selectors パラメータで指定することで、ラベルを使ったフィルタリングができます。同様に、Kubernetes のリソースフィールドの値を field_selectors パラメータで指定することで、フィールドの値を使ったフィルタリングができます。詳細は Kubernetes のドキュメントを参照してください。「app=prometheus」というラベルが付いている Pod のうち、起動状態が Running になっていないものを取得するには List 5-45 のように記述します。

List 5-45　ラベルとフィールドを使用したフィルタリング例: ./Container/k8s_info.yml

```
- name: not runnnig prometheus pods
  kubernetes.core.k8s_info:
    namespace: monitoring
    kind: Pod
```

＊11 kubectl get コマンドの「-A」または「--all-namespaces」オプションに相当

```
    label_selectors:
      - app=prometheus
    field_selectors:
      - status.phase!=Running
```

関　連

▷ kubernetes.core.k8s_info – Describe Kubernetes (K8s) objects － Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/kubernetes/core/k8s_info_module.html

▷ Labels and Selectors | Kubernetes

https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

▷ Field Selectors | Kubernetes

https://kubernetes.io/docs/concepts/overview/working-with-objects/field-selectors/

5-3-3　Pod のログを取得する

キーワード

▷ kubectl logs コマンド

▷ kubernetes.core.k8s_log モジュール

方　法

Pod のログを取得するには kubernetes.core.k8s_log モジュールを使用します。

List 5-46 のプレイブックでは kubernetes.core.k8s_info モジュールを使用して Pod 名とコンテナ名を Ansible 実行時に取得し、その内容をパラメータに指定しています。

List 5-46　kubernetes.core.k8s_log モジュールの利用例: ./Container/k8s_log.yml

```
---
- hosts: k8s
  gather_facts: false

  tasks:
    - name: get pod info
      kubernetes.core.k8s_info:
        namespace: monitoring
        kind: Pod
```

```
10:     register: k8s_info_result
11:
12:   - name: set_fact
13:     ansible.builtin.set_fact:
14:       pod_info:
15:         name: '{{ k8s_info_result.resources
16:                   | selectattr("metadata.name", "match", "prometheus-server")
17:                   | map(attribute="metadata.name")
18:                   | first
19:                 }}'
20:         containers: '{{ k8s_info_result.resources
21:                         | selectattr("metadata.name", "match", "prometheus-server")
22:                         | map(attribute="spec.containers")
23:                         | first
24:                         | map(attribute="name")
25:                         | list
26:                       }}'
27:
28:   - name: get logs from pod
29:     kubernetes.core.k8s_log:
30:       namespace: monitoring
31:       name: "{{ pod_info.name }}"
32:       container: "{{ item }}"
33:     loop: "{{ pod_info.containers }}"
```

解　説

kubernetes.core.k8s_log モジュールの主なパラメータは Table 5-16 のとおりです。

Table 5-16　kubernetes.core.k8s_log モジュールの主なパラメータ

パラメータ名	説明
namespace	対象ネームスペース名
name	対象リソース名
container	対象コンテナ名

`kubectl logs` と同様、1 つの Pod に複数のコンテナを含む場合は、container パラメータにコンテナ名の指定も必要です。

Pod 名は Deployment リソースからデプロイされた Pod のようにランダムな文字列を含む場合は、事前に k8s_info モジュールなどで取得しておく必要がありますが、k8s_info と同様に label_selectors を使って絞り込むこともできます。

List 5-47　ラベルを使ったフィルタリング例: ./Container/k8s_log.yml

```
- name: get logs with label
  kubernetes.core.k8s_log:
    namespace: example
    label_selectors:
      - app=sample-http
```

デフォルトは Pod が対象ですが、Pod 以外のログを参照する場合は kind パラメータを使用して Deployment などのリソースタイプを指定できます。

補　足

CLI の「`kubectl logs -l`」コマンドでセレクタ指定を使用した場合は複数 Pod のログを取得できますが、k8s_log モジュールは該当する Pod のうち 1 つの Pod のログしか取得できないので注意してください。

関　連

▷ kubernetes.core.k8s_log – Fetch logs from Kubernetes resources － Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/kubernetes/core/k8s_log_module.html`

5-3-4　Pod 内のコマンドを実行する

キーワード

▷ `kubectl exec` コマンド
▷ kubernetes.core.k8s_exec モジュール

方　法

Pod 内のコマンドを実行するための `kubectl exec` 相当の操作は、kubernetes.core.k8s_exec モジュールで実行できます。

List 5-48 のプレイブックでは、Prometheus の Pod で「`prometheus --version`」コマンドを実行します。

List 5-48　kubernetes.core.k8s_exec モジュールの利用例: ./Container/k8s_exec.yml

```
---
- hosts: k8s
  gather_facts: false

  tasks:
    - name: get pod info
      kubernetes.core.k8s_info:
        namespace: monitoring
        kind: Pod
      register: result

    - name: set_fact
      ansible.builtin.set_fact:
        pod_info:
          name: '{{ result.resources
                    | selectattr("metadata.name", "match", "prometheus-server")
                    | map(attribute="metadata.name")
                    | first
                  }}'
          containers: prometheus-server

    - name: exec command on prometheus
      kubernetes.core.k8s_exec:
        namespace: monitoring
        pod: "{{ pod_info.name }}"
        container: "{{ pod_info.containers }}"
        command: prometheus --version
```

解　説

kubernetes.core.k8s_exec モジュールの主なパラメータは Table 5-17 のとおりです。

Table 5-17　kubernetes.core.k8s_exec モジュールの主なパラメータ

パラメータ名	説明
namespace	対象ネームスペース名
pod	対象 Pod 名
container	対象コンテナ名
command	実行するコマンド

指定した Pod およびコンテナ内でコマンドを実行します。

ansible.builtin.command モジュールなどと異なり、Pod 上で実行されるコマンドの失敗によってタス

クは失敗とならないので、成否の判定は戻り値の return_code にセットされる値を確認します。

補　足

ansible.builtin.shell モジュールのようにシェルは使っていないので、「`ls > /tmp/ls.txt`」のように指定しても、リダイレクトなどは使用できません。コンテナ内のシェルコマンドを使って、ファイルを出力するのであれば「`sh -c "echo 'abc' > /tmp/abc"`」のように指定します。

関　連

▷ kubernetes.core.k8s_exec – Execute command in Pod － Ansible Documentation
`https://docs.ansible.com/ansible/latest/collections/kubernetes/core/k8s_exec_module.html`

5-3-5　Helm を使ってアプリケーションをデプロイする

キーワード

▷ Helm
▷ `helm install` コマンド
▷ kubernetes.core.helm モジュール

方　法

Helm チャートとして提供されているアプリケーションの Kubernetes クラスタへのデプロイは、kubernetes.core.helm モジュールを使用します。

モニタリングツールの 1 つである Prometheus[*12]をデプロイするプレイブックは List 5-49 のとおりです。

List 5-49　kubernetes.core.helm モジュールの利用例: ./Container/k8s_helm.yml

```
1: ---
2: - hosts: k8s
3:   gather_facts: false
4:
5:   vars:
```

＊ 12　prometheus-community/helm-charts: Prometheus community Helm charts
`https://github.com/prometheus-community/helm-charts`

```
  helm_values_yaml_localpath: helm/prometheus-values.yaml
  helm_values_yaml_remotepath: /var/tmp/kubernetes/helm/prometheus-values.yaml

tasks:
  - name: copy values.yaml for helm
    ansible.builtin.copy:
      src: "{{ helm_values_yaml_localpath }}"
      dest: "{{ helm_values_yaml_remotepath }}"

  - name: delploy prometheus helm chart
    kubernetes.core.helm:
      chart_ref: prometheus-community/prometheus
      release_name: prometheus
      release_namespace: monitoring
      create_namespace: true
      update_repo_cache: true
      release_state: present
      values_files:
        - "{{ helm_values_yaml_remotepath }}"
```

解　説

kubernetes.core.helm モジュールで使用する主なパラメータは Table 5-18 のとおりです。

Table 5-18　kubernetes.core.helm モジュールの主なパラメータ

パラメータ名	説明
chart_ref	チャート名
release_name	チャートをデプロイするリリース名
release_namespace	チャートのインストール先ネームスペース
create_namespace	true 指定でネームスペースを自動作成
update_repo_cache	リポジトリ情報の更新を実施
values_files	チャートのパラメータである values.yaml の指定
release_values	チャートのパラメータをインライン指定

Helm でデプロイするアプリケーションのパラメータは、通常は `values.yaml` などのファイル名の設定値ファイルを用意して使用します。Ansible の kubernetes.core.helm モジュールでは、values_files パラメータに従来どおりの YAML ファイルのパスを指定する方法と、release_values パラメータでタスク内にインラインで設定値ファイルの内容を記述する方法があります。

冒頭の例では、values_files パラメータで設定値ファイルのパスを指定して Prometheus を Kubernetes クラスタへデプロイします。

設定値ファイルは、コントロールノードではなくマネージドノードのファイルシステム上に配置されている必要があるので、プレイブックとは別にリポジトリで管理されているのであれば Git で取得するなどして配置します。本項の例ではコントロールノード上にあるファイルを ansible.builtin.copy モジュールで転送します。

もう 1 つの release_values を使ったタスク内にインラインで設定値を記述する例として、Prometheus などのデータを可視化するツールである Grafana[*13]をデプロイするには、以下のようなタスクを作成します。タスク内に記述できるので、パラメータなどを Jinja2 テンプレートのように変数にすることもできます。List 5-50 の例は PersistentVolume のボリュームサイズを変数にしています。

List 5-50　release_values パラメータを使った設定値の指定: ./Container/k8s_helm.yml

```
- name: deloy grafana helm chart
  vars:
    size: "1Gi"
  kubernetes.core.helm:
    chart_ref: grafana/grafana
    release_name: grafana
    release_namespace: monitoring
    create_namespace: true
    update_repo_cache: true
    release_state: present
    release_values:
      service:
        type: LoadBalancer
      persistence:
        type: pvc
        enabled: true
        accessModes:
          - ReadWriteOnce
        size: "{{ size }}"
```

・release_namespace パラメータは、デプロイ先のネームスペースを指定します。デプロイ先ネームスペースは create_namespace パラメータに true を指定しておけば、存在しなかった場合は自動で作成されます。

・update_repo_cache パラメータは、デプロイ前にローカルのリポジトリ情報を更新します。

・chart_ref にはチャートのリポジトリ名とチャート名を指定します。CLI コマンドで `helm repo install` 実行時に指定するチャート名と同じものです。

＊ 13　Grafana Community Kubernetes Helm Charts | helm-charts
`https://grafana.github.io/helm-charts/`

・release_name にはデプロイするリリース名を指定します。

応　用

release_state に absent を指定すると、release_name に指定したリリースをアンインストールします。

補　足

実行にはマネージドノード上で Helm の CLI コマンドがインストールされている必要があります。Helm の CLI コマンドのインストールについては Helm のインストールドキュメントを参照してください。

また、はじめて使用する場合は Helm のリポジトリ設定が必要なので、kubernetes.core.helm_repository モジュールを使用して設定を行ってください。

関　連

▷ Helm

https://helm.sh/

▷ Helm | Installing Helm

https://helm.sh/docs/intro/install/

▷ kubernetes.core.helm – Manages Kubernetes packages with the Helm package manager — Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/kubernetes/core/helm_module.html

▷ prometheus-community/helm-charts: Prometheus community Helm charts

https://github.com/prometheus-community/helm-charts

▷ Grafana Community Kubernetes Helm Charts | helm-charts

https://grafana.github.io/helm-charts/

5-3-6　Helm リポジトリを設定する

キーワード

▷ helm repo add コマンド

▷ kubernetes.core.helm_repository モジュール

方　法

kubernetes.core.helm_repository モジュールを使って Helm チャートをインストールするためのリポジトリを設定できます。

Prometheus 用の Helm チャートのリポジトリを設定するプレイブックは List 5-51 のとおりです。

List 5-51　kubernetes.core.helm_repository モジュールの利用例: ./Container/k8s_helm.yml

```
---
- hosts: k8s
  gather_facts: false

  tasks:
    - name: add Prometheus repository
      kubernetes.core.helm_repository:
        repo_name: prometheus-community
        repo_url: https://prometheus-community.github.io/helm-charts
```

解　説

Helm チャートのリポジトリ管理を行います。Helm の CLI ツールの `helm repo` に相当します。

使用する主なパラメータは Table 5-19 のとおりです。

Table 5-19　kubernetes.core.helm_repository モジュールの主なパラメータ

パラメータ名	説明
repo_name	登録時に設定する名称
repo_url	リポジトリ URL

repo_url パラメータにチャートの URL を指定し、設定名を repo_name パラメータに指定します。

冒頭の例のプレイブックであれば、Operation 5-1 のコマンドと同等です。

Operation 5-1　helm repo add 実行例

```
$ helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

リポジトリについては、各アプリケーションのドキュメントなどを参照してください。Prometheus の場合は Prometheus Community の GitHub リポジトリ[*14]です。

＊ 14　prometheus-community/helm-charts: Prometheus community Helm charts
https://github.com/prometheus-community/helm-charts

応　用

設定したリポジトリは Helm コマンドと同様に、実行するユーザの$HOME/.config ディレクトリ以下の設定ファイルに記録されます。設定ファイルのパスを変更したい場合は、環境変数 HELM_CONFIG_HOME に設定ファイルのパスを指定すると変更できます。

List 5-52　Helm の設定ファイルのパスを環境変数で指定する例

```
1:    - name: add Prometheus repository
2:      kubernetes.core.helm_repository:
3:        name: prometheus-community
4:        repo_url: https://prometheus-community.github.io/helm-charts
5:      environment:
6:        HELM_CONFIG_HOME: /opt/helm/config
```

デフォルト以外の設定ファイルパスを使用する場合、「List 5-52 Helm の設定ファイルのパスを環境変数で指定する例」のようにタスクで環境変数を設定すると該当タスクでのみ指定の設定ファイルが使用されます。他のタスクの Helm 操作でも任意のファイルパスを使用するには、各タスクで環境変数の指定が必要です。すべてのタスクで一律に設定変更する場合は、タスクではなくプレイの環境変数に指定できます。

指定 URL のリポジトリを削除したい場合は、repo_state パラメータに absent を指定すれば削除されます。

関　連

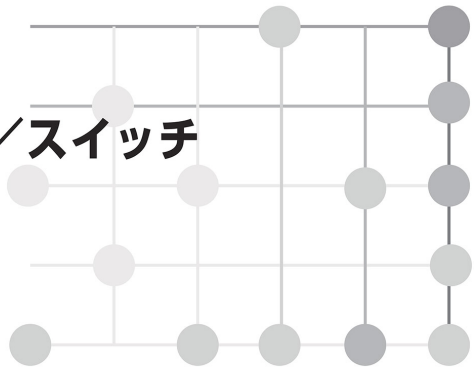
▷ Helm

https://helm.sh/

▷ kubernetes.core.helm_repository - Manage Helm repositories. — Ansible Documentation

https://docs.ansible.com/ansible/latest/collections/kubernetes/core/helm_repository_module.html

第６章
ルータ／スイッチ



Ansible は 50 以上のさまざまなルータやスイッチのプラットフォームに対応しています。show コマンドによる情報取得や、インターフェイスやルーティング、運用管理機能などの設定変更ができます。モジュールを組み合わせることにより、コマンド実行結果のファイルへの保存や、期待値との比較など、作業手順書上の手順を自動化できます。

本章では、ルータ／スイッチ対応の概要と、Cisco IOS のネットワーク機器をマネージドノードとした、情報取得や設定変更のためのプレイブックを紹介します。

6-1　ルータ／スイッチ対応の概要

各プレイブックの紹介に先立ち、Ansible がルータやスイッチに対してできることや環境の準備方法、前提とする構成を説明します。

6-1-1　対応範囲

Ansible が対応するプラットフォームや、自動化できる作業内容を説明します。

■ 対応プラットフォーム

Ansible は Cisco IOS、IOS XR、NX-OS、Juniper Junos、Arista EOS などのプラットフォームに対応しています。各プラットフォーム向けのモジュールは、基本的にコレクション単位で提供されています（Table 6-1）。

Table 6-1　プラットフォームとコレクション名（抜粋）

プラットフォーム名	コレクション名
Cisco IOS	cisco.ios
Cisco IOS XR	cisco.iosxr
Cisco NX-OS	cisco.nxos
Juniper Junos	junipernetworks.junos
Arista EOS	arista.eos
35 のプラットフォーム ‡	community.network
共通	ansible.netcommon

‡ community.network コレクション 3.0.0 現在

Table 6-1 に挙げたコレクションは、本書の前提どおり `pip` コマンドで「`ansible`」をインストールする際（Ansible Community Package）に同梱されるコレクションの一部です。同梱されるコレクションは公式ドキュメントの「Collection Index」[*1]を参照してください。同梱されないコレクションを利用する場合は、Ansible Galaxy[*2]で目的のコレクションを探して「`ansible-galaxy collection install`」コマンドでインストールします。コマンドの詳細は「**1-4-3 コレクションの操作**」を参照してください。

＊1　Collection Index
`https://docs.ansible.com/ansible/latest/collections/index.html`

＊2　Ansible Galaxy
`https://galaxy.ansible.com/`

「ansible.netcommon」コレクション[*3]はプラットフォーム個別ではなく、共通のコレクションです。NETCONF や RESTCONF で設定変更、情報取得するモジュールや、SSH や NETCONF、HTTP API 接続のためのコネクションプラグイン、IP アドレス計算などのネットワーク関連フィルタが含まれます。

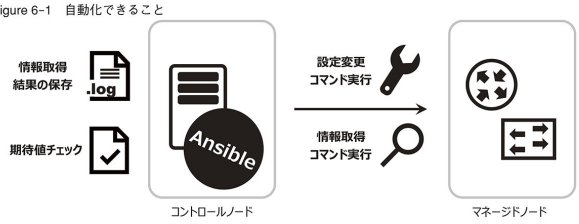
■ 自動化できること

コレクション内に含まれる各モジュールによって、show コマンドによる情報取得や、インターフェイスやルーティング、運用管理機能などの設定ができます。モジュールで指定した各パラメータに従って、コマンドや API リスクエストを生成して実行する仕組みです。

モジュールには、コマンドを直接指定するタイプ（*_command や *_config）や、あるべき状態を宣言的に指定して内部でコマンドを生成するタイプがあります。コマンドを直接指定するタイプでは、Jinja2 テンプレートが利用できるため、テンプレート化したコンフィグと変数を組み合わせて効率的にコマンドを生成できます。

show コマンドの実行結果は変数に格納できます。そのまま、または加工して実行ログやファイルに出力したり、期待した値どおりかのチェックもできます（Figure 6-1）。

Figure 6-1　自動化できること



6-1-2　環境の準備

Cisco IOS のネットワーク機器向けのモジュールを利用するために必要な、コントロールノード側とマネージドノード側の準備について説明します。

＊3　ansible.netcommon コレクション
https://docs.ansible.com/ansible/latest/collections/ansible/netcommon/index.html

■ コントロールノード側の準備

コントロールノードから Cisco IOS のネットワーク機器に SSH 接続するためには、Python パッケージ「paramiko」が必要です。コントロールノード側の「ansible」をインストールした Python 環境に「paramiko」を追加でインストールします。

Operation 6-1　paramiko のインストール実行例

```
$ pip install paramiko
```

モジュール実行時に利用する Python インタープリタの設定も必要です。設定には ansible_python_interpreter 変数か、ansible.cfg の defaults セクションの「interpreter_python」を利用します。たとえば「ansible」を「/home/ansible/myvenv」という venv にインストールした場合は「/home/ansible/myvenv/bin/python」を指定します。ほかには、venv のパスを直接指定するのではなく、マジック変数「ansible_playbook_python」を指定する方法もあります（List 6-1）。

List 6-1　YAML 形式で Python インタープリタを設定する例

```
1: ansible_python_interpreter: "{{ ansible_playbook_python }}"
```

■ マネージドノード（ネットワーク機器）側の準備

マネージドノード側では、SSH 接続を有効化しておきます。なお、本章で紹介するプレイブックは、TELNET による接続には対応していないので注意してください。

6-1-3　cisco.ios コレクションのモジュールの基本仕様

「cisco.ios」コレクションに含まれるモジュールの基本的な仕様について説明します。

■ モジュール一覧と分類

「cisco.ios」コレクションには、Cisco IOS のネットワーク機器に対応する「ios_」から始まる名前のモジュールがあります。モジュールの一覧は公式ドキュメントの「cisco.ios」コレクション詳

細ページ[*4]に掲載されています。

30 以上の多くのモジュールがありますが、分類を意識することで目的のモジュールを見つけやすくなります。大きく分けて情報取得と確認をするモジュールと、設定変更するモジュールに分類できます（Table 6-2）。

Table 6-2　操作によるモジュールの分類

操作分類	設定の指定方法	モジュールの例（すべて cisco.ios コレクション）
情報取得と確認	-	ios_command、ios_facts、ios_ping
設定変更	コマンドを直接指定	ios_config
設定変更	あるべき状態を宣言的に指定	ios_user、ios_interfaces、ios_static_routes

■ 冪等性の担保

設定変更をするモジュールは、実際に設定変更する前に機器の設定を確認して、差分を求めます。そして、差分を埋めるためのコマンドを投入します。差分がなく、投入すべきコマンドがない場合は、タスクの実行ステータスは「ok」になります。

冪等性を担保するために、差分を正しく求められるようにインターフェイス名やコマンドを省略せず指定する必要があります。たとえば、running-config 上のインターフェイスの表記が「GigabitEthernet0/0」の場合は、プレイブック上も同じ表記で指定します。「gi0/0」のように省略して指定してしまうと正しく比較できません。正しく比較できない結果、タスクの実行ステータスが毎回 changed になってしまうので注意してください[*5]。

なお、本章では設定変更のプレイブックと併せて、プレイブックの実行によって投入されるコマンド例も掲載しています。これは、プレイブックで指定する設定が、まだ機器側に入っていない状態を想定しています。

■ 接続に利用する特別な変数

ネットワーク機器への接続や認証、内部の処理方法を決めるために特別な変数を利用します。よく利用する変数は Table 6-3 のとおりです。

＊4　「cisco.ios」コレクション
https://docs.ansible.com/ansible/latest/collections/cisco/ios/index.html

＊5　Ansible 公式 FAQ(Why do the *_config modules always return changed=true with abbreviated commands?)
https://docs.ansible.com/ansible/latest/network/user_guide/faq.html#why-do-the-config-modules-always-return-changed-true-with-abbreviated-commands

Table 6-3　接続に利用する特別な変数

変数名	説明
ansible_connection	利用するコネクションプラグイン。 Cisco IOS では「ansible.netcommon.network_cli」を指定
ansible_port	接続先のポート番号（ansible.netcommon.network_cli コネクションプラグイン利用時のデフォルトは 22）
ansible_network_os	プラットフォーム名。Cisco IOS では「cisco.ios.ios」を指定
ansible_user	ログインに利用するユーザ名
ansible_password	パスワード認証を利用する場合のパスワード
ansible_become	ログイン後に特権モードへ切り替えるかどうか（true、false）。デフォルトは false
ansible_become_method	特権モードに移行する方法。 Cisco IOS では「ansible.netcommon.enable」を指定
ansible_become_password	特権モードに移行する際のパスワード

ログインユーザが特権を持つ場合や、プレイブックで指定する処理に特権が不要な場合は特権モードに関する変数の指定は不要です。具体的には、ansible_become、ansible_become_method、ansible_become_password 変数です。利用する変数の一覧は公式ドキュメントの「IOS Platform Options」[*6]や、「ansible.netcommon.network_cli」コネクションプラグインのページ[*7]を参照してください。

■ ネットワークリソースモジュール

あるべき状態を宣言的に指定するタイプの設定変更モジュールには、ネットワークリソースモジュールという高機能なモジュールがあります。インターフェイスの基本設定であれば「cisco.ios.ios_interfaces」、スタティックルート設定であれば「cisco.ios.ios_static_routes」のように特定の機能ごとにモジュールがあります。config パラメータで設定内容を指定し、state パラメータで「merged」や「deleted」などの状態種別を指定する、という共通の仕様を持ちます。

state パラメータは Table 6-4 の値を指定できます。詳細は公式ドキュメントの「Network Resource Module」[*8]を参照してください。本章のプレイブックでは、デフォルトである「merged」を中心に扱います。

＊6　IOS Platform Options
https://docs.ansible.com/ansible/latest/network/user_guide/platform_ios.html
＊7　「ansible.netcommon.network_cli」コネクションプラグイン
https://docs.ansible.com/ansible/latest/collections/ansible/netcommon/network_cli_connection.html
＊8　Network Resource Module
https://docs.ansible.com/ansible/latest/network/user_guide/network_resource_modules.html

Table 6-4　state パラメータの一覧

state パラメータの値	説明	機器への接続	設定変更
merged	指定した設定をマージした状態にする（デフォルト）	する	する
replaced	指定した設定に置き換えた状態にする	する	する
overridden	指定した設定を上書きし、指定しなかった設定を削除した状態にする。たとえば cisco.ios.ios_interfaces モジュールの config パラメータ内の name パラメータで GigabitEthernet0/0 のみ指定した場合は、GigabitEthernet0/0 の設定は上書きし、他のインターフェイスの基本設定は削除する	する	する
deleted	指定した設定を削除した状態にする	する	する
gathered	機器からコンフィグを収集し、構造化データにパースする	する	しない
rendered	config パラメータで指定した設定を、機器に投入するためのコンフィグに変換する	しない	しない
parsed	running_config パラメータで指定したコンフィグを、config パラメータで扱うための構造化データにパースする	しない	しない

このように state パラメータの値によって挙動が大きく異なるため注意が必要です。とくに「`replaced`」「`overridden`」「`deleted`」は設定の削除を伴うことがます。挙動をよく確認したい場合は、`ansible-playbook` コマンドのチェックモードを利用し、投入予定コマンドをあらかじめ確認することもできます。詳細は「**6-6-4 投入予定のコマンドをチェックモードで確認する**」を参照してください。

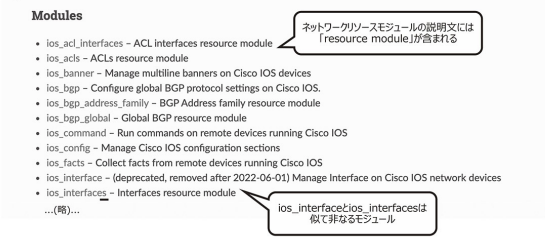
なお、ネットワークリソースモジュールかどうかは、公式ドキュメントの説明文やモジュール名によって区別できます。ネットワークリソースモジュールの説明文[*9]には「ACL interfaces resource module」のように「resource module」が含まれます。モジュール名は、末尾が複数形の「s」か「_global」であるという傾向があります。似たモジュール名でも複数形の「s」が付くモジュールと付かないモジュールがありますが、似て非なるモジュールなので注意してください（Figure 6-2）。

■ タイムアウトのチューニング

実行に時間がかかるコマンドを実行する場合「`command timeout triggered, timeout value is 30 secs.`」のようなエラーでタイムアウトが発生することがあります。デフォルトのコマンドタイムアウトは 30 秒です。コマンド実行タイムアウトは、ansible.netcommon.network_cli コネクションプ

＊9　cisco.ios コレクションのモジュールと説明文の一覧
`https://docs.ansible.com/ansible/latest/collections/cisco/ios/#modules`

Figure 6-2　ネットワークリソースモジュールの説明文



ラグインの persistent_command_timeout[*10]パラメータによって変更できます。変数で指定する場合は「`ansible_command_timeout`」変数で指定します。

`ansible.cfg` で指定する場合は List 6-2 のように指定します。

List 6-2　コマンド実行タイムアウト値を 60 秒に変更

```
1: [persistent_connection]
2: command_timeout = 60
```

その他、接続関連のトラブルシューティング方法は、公式ドキュメントの「Network Debug and Troubleshooting Guide」[*11]に掲載されています。

6-1-4　本章の前提構成

6-2 節以降で掲載する各プレイブックの前提となる環境や、インベントリ、変数定義ファイルについて説明します。

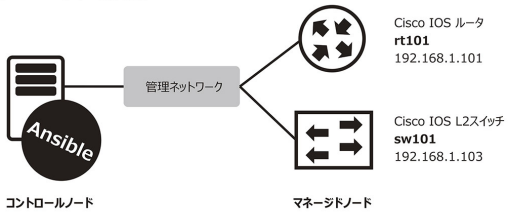
＊ 10　persistent_command_timeout
`https://docs.ansible.com/ansible/latest/collections/ansible/netcommon/network_cli_connection.html#parameter-persistent_command_timeout`

＊ 11　Network Debug and Troubleshooting Guide
`https://docs.ansible.com/ansible/latest/network/user_guide/network_debug_troubleshooting.html`

■ 環境

Cisco IOS のルータとスイッチをマネージドノードとした環境を前提とします。コントロールノードからマネージドノードへは、パスワード認証による SSH 接続ができる状態とします（Figure 6-3）。

Figure 6-3　本章の前提構成



動作確認に利用したネットワーク機器の OS は以下のとおりです。いずれも Cisco Modeling Labs 2.2（Cisco が提供しているネットワークシミュレーションツール）上の仮想ノードです。

- ルータ: Cisco IOS 15.9(3)M3
- L2 スイッチ: Cisco IOS 15.2

主に利用するコレクションは「`cisco.ios`」のバージョン「`2.3.0`」です。本書の前提どおり pip で「`ansible==4.2.0`」をインストールすると同梱されるコレクションです。そのため、とくに追加作業は不要です。コレクションがインストールされていることを確認する場合は「`ansible-galaxy collection list`」コマンドを実行します。

Operation 6-2　インストールされているコレクションの確認

```
$ ansible-galaxy collection list
Collection                    Version
----------------------------- -------
amazon.aws                    1.5.0
ansible.netcommon             2.2.0
ansible.posix                 1.2.0
ansible.utils                 2.3.0
...(略)...
cisco.ios                     2.3.0
```

```
...(略)...
```

■ インベントリと変数定義ファイル

マネージドノードを定義するインベントリファイルは List 6-3 のとおりです。ルータと L2 スイッチがそれぞれグループに所属し、さらに親グループとして ios に所属します。

List 6-3　インベントリ: ./router-switch/inventory.ini

```
[ios:children]
rt
sw

[rt]
rt101 ansible_host=192.168.1.101

[sw]
sw101 ansible_host=192.168.1.103
```

本章のプレイブックでは基本的にはルータを対象とします。動作確認環境においてスイッチでしか動作しないモジュールを利用する場合はスイッチを対象としています。各プレイブックの冒頭で「hosts: rt」と指定しているものがルータ、「hosts: sw」と指定しているものがスイッチを対象とすることを示しています。

接続方式や認証情報を定義する変数定義ファイルは List 6-4 のとおりです。ファイル名は、インベントリファイルで定義したグループ名に合わせて「ios.yml」とします。

List 6-4　グループ変数定義ファイル: ./router-switch/group_vars/ios.yml

```
---
ansible_network_os: cisco.ios.ios
ansible_connection: ansible.netcommon.network_cli
ansible_user: admin
ansible_password: password
ansible_python_interpreter: "{{ ansible_playbook_python }}"
```

ユーザ「admin」は特権を持っている前提です。ユーザ名とパスワードは環境に合わせてください。上記の例では分かりやすさのためにパスワードを平文で指定していますが、実運用では ansible-vault によって暗号化した値を指定することを推奨します。

Column　他のプラットフォームへ応用

本章では Cisco IOS のネットワーク機器をマネージドノードとして扱いますが、前述のとおり Ansible は 50 以上のさまざまなプラットフォームに対応しています。他のプラットフォームに対して Ansible を利用する際のポイントを説明します。

- コレクションとモジュール

  プラットフォームによって利用するコレクションが異なります。本章で取り上げる「`cisco.ios`」のほか、Cisco IOS XR 向けの「`cisco.iosxr`」や Juniper Junos 向けの「`junipernetworks.junos`」などがあります。本書の前提どおり pip で「`ansible`」をインストールする際（Ansible Community Package）に同梱されるコレクションの一覧は、公式ドキュメントの「Collection Index」[*12]で確認できます。

  各コレクションには情報取得や設定をするためのモジュールが用意されています。モジュールの一覧は、公式ドキュメントの「Collection Index」から目的のコレクションの説明ページを参照してください。

- 接続方式

  プラットフォームによって対応する接続方式が異なります。利用する接続方式により、マネージドノード側の設定と利用するコネクションプラグインを合わせます。

  SSH 接続の場合は、本章で扱っているとおり「`ansible.netcommon.network_cli`」コネクションプラグインを利用します。Juniper Junos の機器に対して NETCONF で接続する場合は、マネージドノード側で NETCONF による接続を有効にし、コネクションプラグインには「`ansible.netcommon.netconf`」を利用します。この場合、コントロールノードには NETCONF クライアントの Python 実装である「`ncclient`」を `pip` でインストールする必要があります。

  各プラットフォームに対応するコネクションプラグインの一覧は、公式ドキュメントの「Settings by Platform」[*13]を参照してください。

＊ 12　Collection Index
`https://docs.ansible.com/ansible/latest/collections/index.html`

＊ 13　Settings by Platform
`https://docs.ansible.com/ansible/latest/network/user_guide/platform_index.html#settings-by-platform`

- ansible_network_os 変数

  ansible_network_os 変数には、プラットフォームに応じた値を指定します。たとえば、Cisco IOS XR の場合は「cisco.iosxr.iosxr」、Juniper Junos の場合は「junipernetworks.junos.junos」を指定します。一覧は公式ドキュメントの「Settings by Platform」[*14]を参照してください。

6-2　情報取得と確認

Ansible は Cisco IOS のネットワーク機器に対し、show コマンドを実行して情報を取得できます。show コマンドの実行や、結果のファイル出力、実行結果が期待値のとおりか確認するためのプレイブックを紹介します。

情報取得は設定変更よりリスクを抑えて自動化を始められる作業です。自動化の第一歩としても活用してください。

6-2-1　show コマンド実行結果をログに表示する

キーワード

▷ cisco.ios.ios_command モジュール

方　法

任意の show コマンドを実行するには cisco.ios.ios_command モジュールを利用します。ansible.builtin.debug モジュールと組み合わせると、プレイブックの実行ログに show コマンドの実行結果を表示できます。

複数の show コマンドを実行し、結果をログとして表示するプレイブックは List 6-5 のとおりです。

＊ 14　Settings by Platform
https://docs.ansible.com/ansible/latest/network/user_guide/platform_index.html#settings-by-platform

List 6-5　cisco.ios.ios_command モジュールの利用例: ./router-switch/ios_command_debug.yml

```
---
- hosts: rt
  gather_facts: false

  vars:
    # (1) 実行する show コマンドの定義
    show_commands:
      - show running-config
      - show version
      - show interfaces
      - show ip route

  tasks:
    # (2) show コマンドの実行
    - name: exec show commands
      cisco.ios.ios_command:
        commands: "{{ show_commands }}"
      register: res_show_commands

    # (3) show コマンド結果の表示
    - name: debug show commands
      ansible.builtin.debug:
        msg: "{{ res_show_commands.stdout_lines[ansible_loop.index0] }}"
      loop: "{{ show_commands }}"
      loop_control:
        extended: true
```

解　説

(1) では実行する show コマンドを変数「show_commands」として定義します。ネットワーク機器側のターミナルのページャを無効化する「terminal length 0」コマンドは暗黙的に実行されます。そのため、プレイブック側では明示的に指定する必要はありません。

(2) では show コマンドを実行します。ここでは (1) で show コマンドのリストを定義した変数「show_commands」を指定します。リストを指定することで、複数の show コマンドを実行します。後続のタスクで show コマンドの結果を利用するため、レジスタ変数に格納します。

cisco.ios.ios_command モジュールの主なパラメータは Table 6-5 のとおりです。

Table 6-5　cisco.ios.ios_command モジュールの主なパラメータ

パラメータ名	説明
commands	実行する show コマンドをリストで指定する

(3) では show コマンドの実行結果を 1 行 1 要素のリストとして実行ログに表示します。「res_show_commands」変数には複数の show コマンド実行結果が格納されているため、ループを利用して 1 コマンド分ずつ表示します。

プレイブック実行結果例は Operation 6-3 のとおりです。

Operation 6-3　プレイブック実行結果例

```
PLAY [rt] ***********************************************************************

TASK [exec show commands] *******************************************************
ok: [rt101]

TASK [debug show commands] ******************************************************
ok: [rt101] => (item=show running-config) => {
    "msg": [
        "Building configuration...",
        "",
        "Current configuration : 1796 bytes",
        "!",
        "! Last configuration change at 09:45:52 JST Wed Aug 11 2021 by admin",
        "!",
        "version 15.9",
        "service timestamps debug datetime msec",
        "service timestamps log datetime msec",
        "service password-encryption",
        "!",
        "hostname rt101",

...(略)...
ok: [rt101] => (item=show version) => {
    "msg": [
        "Cisco IOS Software, IOSv Software (VIOS-ADVENTERPRISEK9-M), Version 15
.9(3)M3, RELEASE SOFTWARE (fc1)",
        "Technical Support: http://www.cisco.com/techsupport",
...(略)...
```

応　用

表示形式を 1 行 1 要素のリストではなく、文字列の形式にしたい場合は、(3) のタスクの msg パラメータで指定した「stdout_lines」を「stdout」に変更します。

List 6-6　stdout の指定例: ./router-switch/ios_command_debug_advanced.yml

```
...(略)...
    # (3) show コマンド結果の表示
    - name: debug show commands
      ansible.builtin.debug:
        msg: "{{ res_show_commands.stdout[ansible_loop.index0] }}"  # stdout を指定
      loop: "{{ show_commands }}"
      loop_control:
        extended: true
```

ただし、デフォルトのコールバックプラグインの場合は、改行コードがそのまま「\n」として表示されるためやや読みにくくなります。community.general.yaml コールバックプラグインを利用すると、改行コードが表示上の改行として解釈されて読みやすくなります。community.general.yaml コールバックプラグインを利用するように ansible.cfg で指定するには、List 6-7 のように指定します。

List 6-7　community.general.yaml コールバックプラグインの指定

```
[defaults]
stdout_callback=community.general.yaml
```

community.general.yaml コールバックプラグインを利用する場合のプレイブック実行結果例は、Operation 6-4 のとおりです。「stdout」内の改行コードが実際に改行されて表示されます。

Operation 6-4　stdout と community.general.yaml コールバックプラグインによる表示

```
...(略)...
TASK [debug show commands] ***************************************************
ok: [rt101] => (item=show running-config) =>
  msg: |-
    Building configuration...

    Current configuration : 1796 bytes
    !
    ! Last configuration change at 09:45:52 JST Wed Aug 11 2021 by admin
    !
    version 15.9
    service timestamps debug datetime msec
    service timestamps log datetime msec
    service password-encryption
    !
    hostname ios01
...(略)...
ok: [rt101] => (item=show version) =>
```

```
  msg: |-
    Cisco IOS Software, IOSv Software (VIOS-ADVENTERPRISEK9-M), Version 15.9(3)
M3, RELEASE SOFTWARE (fc1)
    Technical Support: http://www.cisco.com/techsupport
...(略)...
```

関　連

▷ cisco.ios.ios_command モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_command_module.html

▷ コールバックプラグインの指定（DEFAULT_STDOUT_CALLBACK）

https://docs.ansible.com/ansible/latest/reference_appendices/config.html#default-stdout-callback

▷ community.general.yaml コールバックプラグイン

https://docs.ansible.com/ansible/latest/collections/community/general/yaml_callback.html

▷ 2-2-1 実行ログの表示形式を変える

6-2-2　show コマンド実行結果をファイルに保存する

キーワード

▷ cisco.ios.ios_command モジュール

▷ ansible.builtin.copy モジュール

方　法

任意の show コマンドを実行するには cisco.ios.ios_command モジュールを利用します。ansible.builtin.copy モジュールと組み合わせると、実行結果をファイルに保存できます。

複数の show コマンドを実行し、それぞれ別ファイルに保存するプレイブックは List 6-8 のとおりです。

List 6-8　cisco.ios.ios_command モジュールの利用例: ./router-switch/ios_command_file.yml

```
1: ---
2: - hosts: rt
3:   gather_facts: false
```

```
 4:
 5:  vars:
 6:    # (1) 実行する show コマンドの定義
 7:    show_commands:
 8:      - show running-config
 9:      - show version
10:      - show interfaces
11:      - show ip route
12:
13:  tasks:
14:    # (2) ログ保存用ディレクトリの作成
15:    - name: create directory
16:      ansible.builtin.file:
17:        path: "./log/{{ inventory_hostname }}"
18:        state: directory
19:      register: logdir
20:
21:    # (3) show コマンドの実行
22:    - name: exec show commands
23:      cisco.ios.ios_command:
24:        commands: "{{ show_commands }}"
25:      register: res_show_commands
26:
27:    # (4) show コマンド結果のファイル保存
28:    - name: save to file
29:      ansible.builtin.copy:
30:        content: "{{ res_show_commands.stdout[ansible_loop.index0] }}"
31:        dest: "{{ logdir.path }}/{{ item | replace(' ', '_') }}.log"
32:      loop: "{{ show_commands }}"
33:      loop_control:
34:        extended: true
```

解　説

(1) では実行する show コマンドを変数「show_commands」として定義します。ネットワーク機器側のターミナルのページャを無効化する「terminal length 0」コマンドは暗黙的に実行されます。そのため、プレイブック側では明示的に指定する必要はありません。

(2) ではログの保存先ディレクトリを作成します。log ディレクトリ配下にインベントリファイル上のマネージドノード名のディレクトリを作成します。これにより、複数のマネージドノードを対象とした場合でも、マネージドノードごとにログ保存ディレクトリが分かれるため整理できます。

(3) では show コマンドを実行します。cisco.ios.ios_command モジュールの主なパラメータは Table 6-6 のとおりです。

Table 6-6　cisco.ios.ios_command モジュールの主なパラメータ

パラメータ名	説明
commands	実行する show コマンドをリストで指定する

(4) では show コマンド実行結果をファイルに保存します。(1) で定義した show コマンド 1 つにつき 1 ファイルの対応で保存します。ファイル名には、show コマンドに含まれるスペースをアンダーバー（_）に置換した文字列を利用します。保存先のディレクトリ名とファイル名によって、どのマネージドノードで何のコマンドを実行した結果のファイルなのかが分かります。

Operation 6-5　保存された show コマンド実行結果のファイル一覧

```
$ ls -1 log/rt101/
show_interfaces.log
show_ip_route.log
show_running-config.log
show_version.log
```

応　用

(1) の show コマンドを変更することで、保存する show コマンドをカスタマイズできます。

関　連

▷ cisco.ios.ios_command モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_command_module.html

▷ 3-3-1 ディレクトリを作成する

▷ 3-3-4 マネージドノードへファイルをコピーする

6-2-3　show コマンドの結果を構造化データにパースする

キーワード

▷ ansible.utils.cli_parse モジュール

▷ ntc-templates

方　法

Cisco IOS のネットワーク機器では、show コマンドの実行結果が人にとって分かりやすい形式で表

示されます。cisco.ios.ios_command モジュールで show コマンドを実行した場合でも同様です。通常の形式は機械処理には不向きなため、値の抽出や加工には正規表現などを駆使したパースが必要です。

パースする際に便利なものがパーサです。パーサを利用すると、自身で正規表現などを使わずに show コマンドの実行結果を、機械処理に向いた構造化データへパースできます。Ansible がパーサとして利用できるものには ntc-templates、Genie Parser（pyATS の一部）などがあります。

ここでは ntc-templates を利用する方法を紹介します。ntc-templates は、Cisco や Juniper、Arista などさまざまなベンダーのネットワーク機器の show コマンドに対応した、パーサのテンプレート集です。内部的には TextFSM というパーサを利用します。ntc-templates を利用するには、あらかじめコントロールノード側に pip でインストールが必要です。

Operation 6-6　ntc-templates のインストール

```
$ pip install ntc-templates
```

任意の show コマンドの実行とパースを 1 つのタスクで行うには、ansible.utils.cli_parse モジュールを利用します。ansible.builtin.debug モジュールと組み合わせると、パースされた実行結果をプレイブック実行ログに表示できます。

show ip route コマンドを実行し、ntc-templates でパースした結果をログとして表示するプレイブックは List 6-9 のとおりです。

List 6-9　ansible.utils.cli_parse モジュールの利用例: ./router-switch/cli_parse.yml

```
1: ---
2: - hosts: rt
3:   gather_facts: false
4:
5:   tasks:
6:     # (1) show コマンドの実行とパース
7:     - name: exec and parse show command
8:       ansible.utils.cli_parse:
9:         command: show ip route
10:        parser:
11:          name: ansible.netcommon.ntc_templates
12:      register: res_route
13:
14:    # (2) show コマンドのパース結果を表示
15:    - name: debug parsed show commands
16:      ansible.builtin.debug:
17:        msg: "{{ res_route.parsed }}"
```

解　説

(1) では show コマンドを実行し、ntc-templates を利用してパースした結果を取得します。

ansible.utils.cli_parse モジュールの主なパラメータは Table 6-7 のとおりです。

Table 6-7　ansible.utils.cli_parse モジュールの主なパラメータ

パラメータ名		説明
command		実行する show コマンドを指定する
parser		パーサを指定する
	name	パーサの名前を指定する（ansible.netcommon.ntc_templates、ansible.netcommon.pyats など）

(2) ではパースされた show コマンド実行結果を実行ログに表示します。ansible.utils.cli_parse モジュールの実行結果を格納したレジスタ変数「res_route」内の「parsed」にはパース後の結果が格納されます。なお「res_route.stdout」と「res_route.stdout_lines」にはパース前の結果が格納されます。

プレイブック実行結果例は Operation 6-7 のとおりです。show ip route コマンド実行結果の宛先やネクストホップの値が、network、mask、nexthop_ip のような構造化データで表示されます。

Operation 6-7　プレイブック実行結果例

```
PLAY [rt] **********************************************************************

TASK [exec and parse show command] *********************************************
ok: [rt101]

TASK [debug parsed show commands] **********************************************
ok: [rt101] => {
    "msg": [
        {
            "distance": "",
            "mask": "24",
            "metric": "",
            "network": "10.1.1.0",
            "nexthop_if": "GigabitEthernet0/3",
            "nexthop_ip": "",
            "protocol": "C",
            "type": "",
            "uptime": ""
        },
        {
            "distance": "",
            "mask": "32",
```

```
            "metric": "",
            "network": "10.1.1.252",
            "nexthop_if": "GigabitEthernet0/3",
            "nexthop_ip": "",
            "protocol": "L",
            "type": "",
            "uptime": ""
        },
...(略)...
```

応　用

他の show コマンドのパース結果を取得したい場合は、ansible.utils.cli_parse モジュールの command パラメータで指定する show コマンドを変更します。ただし、ntc-templates はネットワーク機器上のすべてのコマンドに対応しているわけではありません。対応しているコマンドを確認するには、ntc-templates をインストールしたディレクトリ配下の「templates」ディレクトリにある「index」という、テンプレートのインデックスファイルを確認します。

Operation 6-8　テンプレートのインデックスファイルの確認例

```
$ cat ~/myvenv/lib/python3.9/site-packages/ntc_templates/templates/index
...(略)...
cisco_ios_show_ip_ospf_neighbor.textfsm, .*, cisco_ios, sh[[ow]] ip ospf nei[[ghbor]]
cisco_ios_show_ip_route_summary.textfsm, .*, cisco_ios, sh[[ow]] ip ro[[ute]] sum[[mary]]
cisco_ios_show_ip_access-lists.textfsm, .*, cisco_ios, sh[[ow]] ip acce[[ss-lists]]
...(略)...
```

テンプレートのインデックスファイルでは、テンプレートファイル名とコマンドが対応付けされています。テンプレートファイル名は、Cisco IOS の場合次のような命名規則です。

cisco_ios_<コマンド名のスペースを_に置換した文字列>.textfsm

「Operation 6-8 テンプレートのインデックスファイルの確認例」からは、Cisco IOS の「show ip ospf neighbor」「show ip route summary」「show ip access-lists」コマンドに対応していることが分かります。

ntc-templates のバージョンアップによって対応コマンドは追加されます。もし目的のコマンドがない場合は、最新バージョンをチェックしてください。また、テンプレートファイルを自作することもできます。

目的のコマンドが対応していない場合の別の対処方法には、パーサを変えるという手段もあります。たとえば、ansible.utils.cli_parse モジュールで利用するパーサとして「ansible.netcommon.pyats」を

指定すると、pyATS 内の Genie Parser を利用できます。Genie Parser は Cisco のネットワーク機器を中心に対応しているパーサです。

関　連

▷ ansible.utils.cli_parse モジュール

https://docs.ansible.com/ansible/latest/collections/ansible/utils/cli_parse_module.html

▷ Parsing semi-structured text with Ansible（各種パーサの利用例）

https://docs.ansible.com/ansible/latest/network/user_guide/cli_parsing.html

▷ ntc-templates

https://github.com/networktocode/ntc-templates

▷ Genie Parsers

https://pubhub.devnetcloud.com/media/genie-feature-browser/docs/#/parsers

6-2-4　ファクトを収集する

キーワード

▷ cisco.ios.ios_facts モジュール

方　法

ネットワーク OS のバージョンやシリアルナンバー、インターフェイス情報などのシステム情報はファクトとして収集できます。ファクトを収集するには、gather_facts ディレクティブで「true」を指定（デフォルト）する方法と、cisco.ios.ios_facts モジュールを利用する方法があります。収集できる内容は両者とも同じです[*15]。gather_facts ディレクティブを利用する場合は、各プレイの冒頭で 1 度だけ収集します。一方、cisco.ios.ios_facts モジュールを利用する場合は、タスクとして任意のタイミングで収集できます。ここでは柔軟で応用がききやすい、cisco.ios.ios_facts モジュールを利用したファクトの収集例を紹介します。

ハードウェアとインターフェイスの L3 設定に関するファクトを収集し、ログとして表示するプレイブックは List 6-10 のとおりです。

＊ 15　厳密には gather_facts ディレクティブによる収集のみ「_ansible_facts_gathered: true」という値が含まれます。

List 6-10　cisco.ios.ios_facts モジュールの利用例: ./router-switch/ios_facts.yml

```
---
- hosts: rt
  gather_facts: false    # モジュールで収集するためここでは無効化

  tasks:
    # 指定したファクトを収集
    - name: gather facts
      cisco.ios.ios_facts:
        gather_subset:
          - hardware  # ハードウェアに関するファクトを収集
        gather_network_resources:
          - l3_interfaces    # L3 に関するファクト収集

    # ファクトを表示
    - name: debug facts
      ansible.builtin.debug:
        msg: "{{ ansible_facts }}"
```

解　説

cisco.ios.ios_facts モジュールの主なパラメータは Table 6-8 のとおりです。

Table 6-8　cisco.ios.ios_facts モジュールの主なパラメータ

パラメータ名	説明
gather_subset	収集するファクトのサブセットをリストで指定する（all、min、hardware、config、interfaces）。この指定により収集したファクトは「ansible_facts」という変数名に格納される。「!config」のように最初に「!」を付けると除外できる
gather_network_resources	収集するファクトのリソースをリストで指定する （all、l3_interfaces、static_routes など）。この指定により収集したファクトは「ansible_facts.network_resources」配下に格納される。「!vlans」のように最初に「!」を付けると除外できる

プレイブック実行結果例は Operation 6-9 のとおりです。ファイルシステムなどのハードウェアに関する情報と、インターフェイスの IP アドレスが表示されます。

Operation 6-9　プレイブック実行結果例

```
PLAY [rt] **********************************************************************

TASK [gather facts] ************************************************************
ok: [rt101]
```

```
TASK [debug facts] *************************************************************
ok: [rt101] => {
    "msg": {
        "net_api": "cliconf",
        "net_filesystems": [
            "flash0:"
        ],
        "net_filesystems_info": {
            "flash0:": {
                "spacefree_kb": 1940908.0,
                "spacetotal_kb": 2092496.0
            }
        },
...(略)...
        "net_system": "ios",
        "net_version": "15.9(3)M3",
        "network_resources": {
            "l3_interfaces": [
                {
                    "name": "loopback0"
                },
                {
                    "ipv4": [
                        {
                            "address": "172.16.1.1 255.255.255.252"
                        }
                    ],
                    "name": "GigabitEthernet0/0"
                },
...(略)...
```

応　用

通常の show コマンド実行結果とは異なり、ファクトは構造化データです。そのため、値の抽出や比較の処理が容易です。たとえば、ネットワーク OS のバージョンのみ抽出したい場合は「ansible_facts.net_version」のように指定します。

List 6-11　ファクト内のバージョンのみを表示する例: ./router-switch/ios_facts_advanced.yml

```
1: ...(略)...
2:     # バージョン表示
3:     - name: debug version
4:       ansible.builtin.debug:
5:         msg: "{{ ansible_facts.net_version }}"   # 表示例 "15.9(3)M3"
```

関　連

▷ cisco.ios.ios_facts モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_facts_module.html

6-2-5　show コマンド実行結果が期待どおりかチェックする

キーワード

▷ ansible.builtin.assert モジュール

方　法

ある値が期待どおりか否かをチェックするには、ansible.builtin.assert モジュールを利用します。show コマンドの実行結果に対して利用すると、作業手順書に登場するような show コマンド結果の確認作業を自動化できます。

「show ip route」コマンド実行結果において、特定のネットワークへのネクストホップ IP アドレスが、期待どおりかチェックするプレイブックは List 6-12 のとおりです。

List 6-12　ansible.builtin.assert モジュールの利用例: ./router-switch/assert.yml

```
---
- hosts: rt
  gather_facts: false

  vars:
    #(1) 期待するルートを定義
    expected:
      network: 10.1.11.0
      mask: "24"
      nexthop_ip: 10.1.1.1

  tasks:
    #(2) show コマンドの実行とパース
    - name: exec and parse show command
      ansible.utils.cli_parse:
        command: show ip route
        parser:
          name: ansible.netcommon.ntc_templates
      register: res_route

    #(3) show コマンドの結果が期待する値と一致しているか比較
    - name: assert nexthop
```

```
23:      ansible.builtin.assert:
24:        that:
25:          - (filtered_mask | length) == 1
26:          - filtered_mask[0].nexthop_ip == expected.nexthop_ip
27:      vars:
28:        filtered_network: |-
29:          {{ res_route.parsed | selectattr('network', '==', expected.network) ⇒
30: | list }}
31:        filtered_mask: |-
32:          {{ filtered_network | selectattr('mask', '==', expected.mask) | list }}
```

解　説

(1) では期待するルートを変数で定義します。後のタスクで、抽出や比較のために利用します。

(2) では ansible.utils.cli_parse モジュールで「`show ip route`」コマンドを実行し、パースした結果を取得します。パースすることにより、後のタスクで抽出や加工などの処理がしやすくなります。

(3) では `vars` ディレクティブで対象の宛先とマスクのルート情報を抽出し、ansible.builtin.assert モジュールの that パラメータで、ネクストホップ IP アドレスが期待どおりであることを確認します。

ansible.builtin.assert モジュールの主なパラメータは Table 6-9 のとおりです。

Table 6-9　ansible.builtin.assert モジュールの主なパラメータ

パラメータ名	説明
that	期待する条件式を指定する。リストで複数指定する場合は AND 扱い

プレイブック実行結果例は Operation 6-10 のとおりです。期待する条件式を満たし、タスクの実行ステータスが「ok」となります。

Operation 6-10　プレイブック実行結果例（期待どおりの場合）

```
PLAY [rt] *********************************************************************

TASK [exec and parse show command] ********************************************
ok: [rt101]

TASK [assert nexthop] *********************************************************
ok: [rt101] => {
    "changed": false,
    "msg": "All assertions passed"
}

PLAY RECAP ********************************************************************
```

```
rt101 : ok=2  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

一方、期待する条件式を満たさない場合はエラーになります。

Operation 6-11　期待する条件式を満たさない場合の表示例

```
...(略)...
TASK [assert nexthop] ***********************************************************
fatal: [rt101]: FAILED! => {
    "assertion": "filtered_mask[0].nexthop_ip == expected.nextho_ip",
    "changed": false,
    "evaluated_to": false,
    "msg": "Assertion failed"
}
...(略)...
```

関　連

▷ ansible.builtin.assert モジュール

https://docs.ansible.com/ansible/latest/collections/ansible/builtin/assert_module.html

▷ ansible.utils.cli_parse モジュール

https://docs.ansible.com/ansible/latest/collections/ansible/utils/cli_parse_module.html

▷ Parsing semi-structured text with Ansible（各種パーサの利用例）

https://docs.ansible.com/ansible/latest/network/user_guide/cli_parsing.html

▷ ntc-templates

https://github.com/networktocode/ntc-templates

▷ 2-4-1 期待値チェックをする

6-2-6　ネットワーク機器から ping を実行する

キーワード

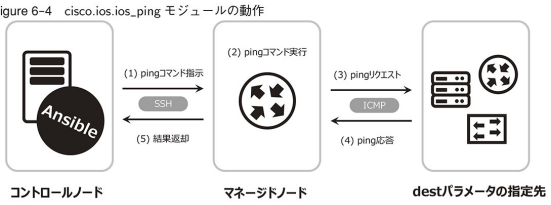
▷ cisco.ios.ios_ping モジュール

方　法

Cisco IOS のネットワーク機器から ping を実行するには cisco.ios.ios_ping モジュールを利用します。cisco.ios.ios_ping モジュールは、マネージドノードである Cisco IOS のネットワーク機器から、モジュール内で指定する宛先に対して、ICMP による ping を実行します。内部的には ping コマンドを生成し

て、Cisco IOS のネットワーク機器上で実行します（Figure 6-4）。ansible.builtin.ping モジュールとは利用するプロトコルも通信区間も異なりますので注意してください。

Figure 6-4　cisco.ios.ios_ping モジュールの動作



ping を実行し、結果をログとして表示するプレイブックは List 6-13 のとおりです。

List 6-13　cisco.ios.ios_ping モジュールの利用例: ./router-switch/ios_ping.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    # ping の実行
    - name: execute ping
      cisco.ios.ios_ping:
        dest: 10.1.1.1
        count: 10
        state: present
      register: res_ping

    # ping 実行結果を出力
    - name: debug ping result
      ansible.builtin.debug:
        msg:
          packet_loss: "{{ res_ping.packet_loss }}"
          packets_rx: "{{ res_ping.packets_rx }}"
          packets_tx: "{{ res_ping.packets_tx }}"
          rtt: "{{ res_ping.rtt }}"
```

解　説

cisco.ios.ios_ping モジュールの主なパラメータは Table 6-10 のとおりです。

Table 6-10　cisco.ios.ios_ping モジュールの主なパラメータ

パラメータ名	説明
dest	ping の宛先の IP アドレスまたはホスト名を指定する
count	送信するパケット数を指定する
source	送信元インターフェイスを指定する
state	成功すべき（present）か、失敗すべき（absent）か指定する。デフォルトは present

プレイブック実行結果例は Operation 6-12 のとおりです。ping コマンドが実行され、パケットロス率や、送受信パケット数、RTT（Round Trip Time）といった実行結果が表示されます。

Operation 6-12　プレイブック実行結果例

```
PLAY [rt] **********************************************************************

TASK [execute ping] ************************************************************
ok: [rt101]

TASK [debug ping result] *******************************************************
ok: [rt101] => {
    "msg": {
        "packet_loss": "0%",
        "packets_rx": "10",
        "packets_tx": "10",
        "rtt": {
            "avg": 4,
            "max": 6,
            "min": 3
        }
    }
}

PLAY RECAP *********************************************************************
rt101 : ok=2  changed=0  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

state パラメータについて補足します。「present」を指定した場合、ロス率が 100%でなければタスクの実行ステータスが「ok」になります。つまり 1 回でも成功すれば「ok」です。一方「absent」を指定した場合、ロス率が 100%の場合のみ「ok」になります。1 回でも成功すれば「failed」です。

応　用

ロス率 0%のみを「ok」とし、ロス率 0%以外を「failed」としたい場合は、failed_when ディレクティブで条件を指定します。

List 6-14　ロス率 0%のみを ok にする例: ./router-switch/ios_ping_advanced.yml

```
1: ---
2: - hosts: rt
3:   gather_facts: false
4:
5:   tasks:
6:     # ping の実行
7:     - name: execute ping
8:       cisco.ios.ios_ping:
9:         dest: 10.1.1.1
10:        count: 10
11:        state: present
12:      register: res_ping
13:      failed_when: res_ping.packet_loss != "0%"    # fail 条件を指定
```

関　連

▷ cisco.ios.ios_ping モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_ping_module.html

6-2-7　設定変更前後のコンフィグ差分を表示する

キーワード

▷ cisco.ios.ios_command モジュール

▷ ansible.utils.fact_diff モジュール

方　法

ansible.utils.fact_diff モジュールを利用すると、2 つ値の差分を取得できます。コンフィグやインターフェイスの状態などの各種 show コマンド実行結果の比較に利用できます。

設定変更前後の「show running-config」コマンドの結果を取得し、差分をログとして表示するプレイブックは List 6-15 のとおりです。

List 6-15　設定変更前後のコンフィグ差分を表示するプレイブック例: ./router-switch/fact_diff.yml

```
1: ---
2: - hosts: rt
3:   gather_facts: false
4:
```

```
 5:  tasks:
 6:    # (1) 設定変更前の running-config 内容の保存
 7:    - name: show running-config (before)
 8:      cisco.ios.ios_command:
 9:        commands:
10:          - show running-config
11:      register: res_before_running_config
12:
13:    # (2) インターフェイスの description の変更
14:    - name: set description
15:      cisco.ios.ios_interfaces:
16:        config:
17:          - name: GigabitEthernet0/3
18:            description: after_description
19:
20:    # (3) 設定変更後の running-config 内容の保存
21:    - name: show running-config (after)
22:      cisco.ios.ios_command:
23:        commands:
24:          - show running-config
25:      register: res_after_running_config
26:
27:    # (4) diff の表示
28:    - name: show diff
29:      ansible.utils.fact_diff:
30:        before: "{{ res_before_running_config.stdout[0] }}"
31:        after: "{{ res_after_running_config.stdout[0] }}"
32:        plugin:
33:          vars:
34:            skip_lines:
35:              - "^Current configuration.+$"
36:              - "! Last configuration change at .+$"
```

解　説

(1) では設定変更前の running-config の内容をレジスタ変数に保存します。

(2) ではインターフェイスの description を変更します。変更内容はあくまでサンプルなので、必要に応じて差し替えてください。

(3) では設定変更後の running-config の内容をレジスタ変数に保存します。

(4) では設定変更前後の running-config の差分を表示します。ansible.utils.fact_diff モジュールの主なパラメータは Table 6-11 のとおりです。

Table 6-11　ansible.utils.fact_diff モジュールの主なパラメータ

パラメータ名			説明
before			比較対象にする 1 番目の値を指定する
after			比較対象にする 2 番目の値を指定する
plugin			diff プラグインを指定する
	name		diff プラグインの名前を指定する（デフォルトは ansible.utils.native）
	vars		diff プラグインのためのパラメータを指定する
		skip_lines	比較対象外にする行を正規表現のリストで指定する

skip_lines パラメータでは、コンフィグのサイズや保存日時のような、コンフィグ内容そのものではない行の差分を比較対象外として指定をしています。これにより、コンフィグ内容そのものに差分があるときのみ差分が発生するように調整しています。

プレイブック実行結果例は Operation 6-13 のとおりです。インターフェイスの description の変更が差分として表示されます。行頭が「-」の行は削除された行、「+」の行は追加された行です[*16]。差分がある場合はタスクの実行ステータスが「changed」、ない場合は「ok」となります。

Operation 6-13　プレイブック実行結果例

```
PLAY [rt] **********************************************************************

TASK [show running-config (before)] ********************************************
ok: [rt101]

TASK [set description] *********************************************************
changed: [rt101]

TASK [show running-config (after)] *********************************************
ok: [rt101]

TASK [show diff] ***************************************************************
--- before
+++ after
@@ -84,7 +84,7 @@
  media-type rj45
 !
 interface GigabitEthernet0/3
- description before_description
+ description after_description
```

＊16　unified diff 形式

```
  ip address 10.1.1.252 255.255.255.0
  duplex auto
  speed auto

changed: [rt101]

PLAY RECAP *********************************************************************
rt101 : ok=4  changed=2  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

応　用

ansible.utils.fact_diff モジュールは、running-config だけでなくその他の各種 show コマンドの結果や、ネストされた変数も比較できます。必要に応じて before や after パラメータの内容を変更してください。

関　連

▷ ansible.utils.fact_diff モジュール

https://docs.ansible.com/ansible/latest/collections/ansible/utils/fact_diff_module.html

▷ 6-2-1 show コマンド実行結果をログに表示する

▷ 6-3-1 インターフェイスの基本設定をする

6-3　インターフェイス設定

インターフェイスの有効化や無効化、VLAN 割り当て、IP アドレスなどインターフェイスに関する設定のためのプレイブックを紹介します。

6-3-1　インターフェイスの基本設定をする

キーワード

▷ cisco.ios.ios_interfaces モジュール

方　法

インターフェイスの有効化、無効化やディスクリプションなどの基本設定には、cisco.ios.ios_interfaces モジュールを利用します。

インターフェイスを有効化し、ディスクリプションや MTU などの基本設定をするプレイブックは List 6-16 のとおりです。

List 6-16　cisco.ios.ios_interfaces モジュールの利用例: ./router-switch/ios_interfaces.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    # インターフェイスの基本設定
    - name: set basic interface configuration
      cisco.ios.ios_interfaces:
        config:
          - name: GigabitEthernet0/4
            enabled: true
          - name: GigabitEthernet0/5
            description: to_rt09
            enabled: true
            mtu: 1454
            speed: 100
            duplex: full
        state: merged
```

解　説

cisco.ios.ios_interfaces モジュールの主なパラメータは Table 6-12 のとおりです。

Table 6-12　cisco.ios.ios_interfaces モジュールの主なパラメータ

パラメータ名		説明
config		設定内容をリストで指定する
	name	設定対象のインターフェイス名を指定する
	description	ディスクリプションを指定する
	enabled	有効化、無効化を指定する。デフォルトは true（有効化）。ネットワーク機器側のデフォルトと異なるとトラブルのもとになり得るため、明示的な指定を推奨
	mtu	MTU を指定する
	speed	リンク速度を指定する
	duplex	全二重、半二重を指定する（full、half、auto）
state		config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定したインターフェイスの設定をマージする

前掲のプレイブックの場合、Operation 6-14 のコマンドを投入します。

Operation 6-14　コマンド投入例

```
interface GigabitEthernet0/4
no shutdown
interface GigabitEthernet0/5
description to_rt09
mtu 1454
speed 100
duplex full
no shutdown
```

応　用

プレイブック内で指定したインターフェイスにおいて、指定した基本設定のみを残して他の基本設定を削除したい場合は、state パラメータで「replaced」を指定します。たとえば List 6-17 のプレイブックの場合、GigabitEthernet0/5 で指定しない mtu、speed、duplex については機器のデフォルト値に戻ります。

List 6-17　state で replaced を指定する例: ./router-switch/ios_interfaces_advanced.yml

```
1: ---
2: - hosts: rt
3:   gather_facts: false
4:
5:   tasks:
6:     # インターフェイスの基本設定
7:     - name: set basic interface configuration by replaced
8:       cisco.ios.ios_interfaces:
9:         config:
10:          - name: GigabitEthernet0/5
11:            description: to_rt09
12:            enabled: true
13:        state: replaced
```

なお、state パラメータで「replaced」を指定した場合、影響範囲は指定したインターフェイスのみです。そのため、この場合 GigabitEthernet0/5 以外のインターフェイスの設定は変更しません。

関　連

▷ cisco.ios.ios_interfaces モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_interfaces_module.html

6-3-2　VLAN を設定する

キーワード

▷ cisco.ios.ios_vlans モジュール

方　法

VLAN の作成や VLAN 名を設定するには、cisco.ios.ios_vlans モジュールを利用します。

VLAN ID「10」と「20」の VLAN を作成するプレイブックは List 6-18 のとおりです。

List 6-18　cisco.ios.ios_vlans モジュールの利用例: ./router-switch/ios_vlans.yml

```
---
- hosts: sw
  gather_facts: false

  tasks:
    # VLAN 設定
    - name: set vlan
      cisco.ios.ios_vlans:
        config:
          - vlan_id: 10
            name: vlan10
          - vlan_id: 20
            name: vlan20
            state: suspend   # suspend 状態にする
            shutdown: enabled   # shutdown 状態にする
        state: merged
```

解　説

cisco.ios.ios_vlans モジュールの主なパラメータは Table 6-13 のとおりです。

Table 6-13　cisco.ios.ios_vlans モジュールの主なパラメータ

パラメータ名		説明
config		設定内容をリストで指定する
	vlan_id	VLAN ID を指定する
	name	VLAN 名を指定する
	mtu	MTU を指定する
	state	VLAN の状態を指定する（active、suspend）
	shutdown	シャットダウンするかどうかを指定する（enabled、disabled）

state	config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定した VLAN の設定をマージする

前掲のプレイブックの場合、Operation 6-15 のコマンドを投入します。

Operation 6-15　コマンド投入例

```
vlan 10
name vlan10
vlan 20
name vlan20
state suspend
shutdown
```

関　連

▷ cisco.ios.ios_vlans モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_vlans_module.html

▷ 6-3-3 インターフェイスの L2 設定をする

6-3-3　インターフェイスの L2 設定をする

キーワード

▷ cisco.ios.ios_l2_interfaces モジュール

▷ アクセス VLAN

▷ トランク VLAN

方　法

インターフェイスへの VLAN 割り当てなどの L2 の設定をするには、cisco.ios.ios_l2_interfaces モジュールを利用します。

GigabitEthernet1/1 にアクセス VLAN、GigabitEthernet1/2 にトランク VLAN を設定するプレイブックは List 6-19 のとおりです。

List 6-19　cisco.ios.ios_l2_interfaces モジュールの利用例: ./router-switch/ios_l2_interfaces.yml

```
1: ---
2: - hosts: sw
```

```
 3:  gather_facts: false
 4:
 5:  tasks:
 6:    # インターフェイスへの VLAN 適用
 7:    - name: apply vlan to interfaces
 8:      cisco.ios.ios_l2_interfaces:
 9:        config:
10:          - name: GigabitEthernet1/1
11:            mode: access
12:            access:
13:              vlan: 10
14:          - name: GigabitEthernet1/2
15:            mode: trunk
16:            trunk:
17:              allowed_vlans: 10-20,30
18:              encapsulation: dot1q
19:        state: merged
```

解　説

cisco.ios.ios_l2_interfaces モジュールの主なパラメータは Table 6-14 のとおりです。

Table 6-14　cisco.ios.ios_l2_interfaces モジュールの主なパラメータ

パラメータ名			説明
config			設定内容をリストで指定する
	name		インターフェイス名を指定する
	mode		switchport のモードを指定する（access、trunk）
	access		アクセス VLAN を指定する
		vlan	アクセス VLAN として割り当てる VLAN ID を指定する
	trunk		トランク VLAN を指定する
		allowed_vlans	トランク VLAN として割り当てる VLAN ID を指定する
		encapsulation	トランク VLAN のカプセル化方式を指定する（dot1q、isl、negotiate）
state			config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定した L2 の設定をマージする

前掲のプレイブックの場合、Operation 6-16 のコマンドを投入します。

Operation 6-16　コマンド投入例

```
interface GigabitEthernet1/1
switchport access vlan 10
switchport mode access
interface GigabitEthernet1/2
switchport trunk encapsulation dot1q
switchport trunk allowed vlan 10-20,30
switchport mode trunk
```

なお、「state: merged」と allowed_vlans パラメータを併用する場合、対象のインターフェイスにすでに「switchport trunk allowed vlan」が設定されているかどうかにより、投入するコマンドが異なります。「switchport trunk allowed vlan」が設定されていない場合は、「Operation 6-16 コマンド投入例」のように allowed_vlans パラメータで指定された VLAN 分の「switchport trunk allowed vlan」コマンドを投入します。

もしすでに「switchport trunk allowed vlan」が設定されている場合は、差分の「switchport trunk allowed vlan add」コマンドを投入します。たとえば、すでに「switchport trunk allowed vlan 10-20」が設定されているインターフェイスに対して「allowed_vlans: 10-20,30」と指定した場合は「switchport trunk allowed vlan add 30」コマンドを投入します。「add」が含まれる点に注目してください。

応　用

対象のインターフェイスにすでに「switchport trunk allowed vlan」が設定されているかどうかにかかわらず、allowed_vlans パラメータで指定したトランク VLAN に置き換える場合は、state パラメータで「replaced」を指定します。たとえば List 6-20 のプレイブックの場合、もしすでに「switchport trunk allowed vlan 10-20,30」が設定されている状態でも、プレイブック実行後は「switchport trunk allowed vlan 30」になります。 結果、VLAN ID 10-20 のトランク VLAN の割り当ては解除されます。

List 6-20　state で replaced を指定する例: ./router-switch/ios_l2_interfaces_advanced.yml

```
1: ---
2: - hosts: sw
3:   gather_facts: false
4:
5:   tasks:
6:     # インターフェイスへの VLAN 適用（replaced）
```

```
 7:     - name: apply vlan to interfaces
 8:       cisco.ios.ios_l2_interfaces:
 9:         config:
10:           - name: GigabitEthernet1/2
11:             mode: trunk
12:             trunk:
13:               allowed_vlans: 30
14:               encapsulation: dot1q
15:         state: replaced
```

関　連

▷ cisco.ios.ios_l2_interfaces モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_l2_interfaces_module.html

▷ 6-3-2 VLAN を設定する

6-3-4　インターフェイスの IP アドレス設定をする

キーワード

▷ cisco.ios.ios_l3_interfaces モジュール

▷ IP アドレス

方　法

インターフェイスに IP アドレスを設定するには、cisco.ios.ios_l3_interfaces モジュールを利用します。

GigabitEthernet0/4 に IPv4 アドレスを、GigabitEthernet0/5 に IPv4、IPv6 アドレスを設定するプレイブックは List 6-21 のとおりです。

List 6-21　cisco.ios.ios_l3_interfaces モジュールの利用例: ./router-switch/ios_l3_interfaces.yml

```
1: ---
2: - hosts: rt
3:   gather_facts: false
4:
5:   tasks:
6:     # IP アドレスの設定
7:     - name: set ip address
8:       cisco.ios.ios_l3_interfaces:
```

```
 9:         config:
10:           - name: GigabitEthernet0/4
11:             ipv4:
12:               - address: 172.31.1.1/24
13:           - name: GigabitEthernet0/5
14:             ipv4:
15:               - address: 172.31.2.1/24
16:             ipv6:
17:               - address: 2001:db8::1/64
18:         state: merged
```

解　説

cisco.ios.ios_l3_interfaces モジュールの主なパラメータは Table 6-15 のとおりです。

Table 6-15　cisco.ios.ios_l3_interfaces モジュールの主なパラメータ

パラメータ名			説明
config			設定内容をリストで指定する
	name		インターフェイス名を指定する
	ipv4		IPv4 関連の設定をリストで指定する
		address	IPv4 アドレスを指定する
	ipv6		IPv6 関連の設定をリストで指定する
		address	IPv6 アドレスを指定する
state			config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定した IP アドレスの設定をマージする

前掲のプレイブックの場合、Operation 6-17 のコマンドを投入します。

Operation 6-17　コマンド投入例

```
interface GigabitEthernet0/4
ip address 172.31.1.1 255.255.255.0
interface GigabitEthernet0/5
ip address 172.31.2.1 255.255.255.0
ipv6 address 2001:db8::1/64
```

なお、すでにインターフェイスにアドレスが設定されている場合、state パラメータが「merged」の場合は IPv4 アドレスと IPv6 アドレスは設定方法が異なります。IPv4 アドレスはプレイブック内で指定した IPv4 アドレスのみに差し替えて設定します。一方、IPv6 アドレスは、設定済み IPv6 アドレスに加えて、追加で設定します。

応　用

IPv6 アドレスを設定する場合の応用を紹介します。もし、指定した IPv6 アドレスのみに差し替えたい場合は state パラメータで「replaced」を指定します。たとえば List 6-22 のプレイブックの場合、すでに GigabitEthernet0/5 に IPv6 アドレスが設定されている状態でも、プレイブックを実行すると IPv6 アドレスは「2001:db8::2/64」のみになります。

List 6-22　state で replaced を指定する例: ./router-switch/ios_l3_interfaces_advanced.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    # IP アドレスの設定
    - name: set ip address
      cisco.ios.ios_l3_interfaces:
        config:
          - name: GigabitEthernet0/5
            ipv4:
              - address: 172.31.2.1/24
            ipv6:
              - address: 2001:db8::2/64
        state: replaced
```

関　連

▷ cisco.ios.ios_l3_interfaces モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_l3_interfaces_module.html

6-4　ルーティング設定

スタティックルーティングや、OSPFv2、BGP の設定をするプレイブックを紹介します。

6-4-1　スタティックルートを設定する

キーワード

▷ cisco.ios.ios_static_routes モジュール

方　法

スタティックルートを設定するには、cisco.ios.ios_static_routes モジュールを利用します。

宛先ネットワーク 10.1.21.0/24 と 10.1.22.0/24 へのスタティックルートを設定するプレイブックは List 6-23 のとおりです。

List 6-23　cisco.ios.ios_static_routes モジュールの利用例: ./router-switch/ios_static_routes.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    # スタティックルートの設定
    - name: set static route
      cisco.ios.ios_static_routes:
        config:
          - address_families:
              - afi: ipv4
                routes:
                  - dest: 10.1.21.0/24
                    next_hops:
                      - forward_router_address: 10.1.2.1
                  - dest: 10.1.22.0/24
                    next_hops:
                      - forward_router_address: 10.1.2.1
        state: merged
```

解　説

cisco.ios.ios_static_routes モジュールの主なパラメータは Table 6-16 のとおりです。

Table 6-16　cisco.ios.ios_static_routes モジュールの主なパラメータ

パラメータ名					説明
config					設定内容をリストで指定する
	address_families				スタティックルートに利用するアドレスファミリをリストで指定する
		afi			Address Family Identifier を指定する（ipv4、ipv6）
		routes			ルートをリストで指定する
			dest		宛先ネットワークを指定する
			next_hops		ネクストホップをリストで指定する
				forward_router_address	ネクストホップのアドレスを指定する

state	config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定したスタティックルートの設定をマージする

前掲のプレイブックの場合、Operation 6-18 のコマンドを投入します。

Operation 6-18　コマンド投入例

```
ip route 10.1.21.0 255.255.255.0 10.1.2.1
ip route 10.1.22.0 255.255.255.0 10.1.2.1
```

応　用

プレイブック内で指定したスタティックルートのみを残して、他のスタティックルートを削除したい場合は、state パラメータで「overridden」を指定します。たとえば「ip route 10.1.29.0 255.255.255.0 10.1.2.1」というスタティックルートが設定されている状態で、前掲のプレイブックの state パラメータで「overridden」を指定して実行すると、もともと設定されていた「ip route 10.1.29.0 255.255.255.0 10.1.2.1」は削除されます。

List 6-24　state で overridden を指定する例: ./router-switch/ios_static_routes_advanced.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    # スタティックルートの設定
    - name: set static route
      cisco.ios.ios_static_routes:
        config:
          - address_families:
              - afi: ipv4
                routes:
                  - dest: 10.1.21.0/24
                    next_hops:
                      - forward_router_address: 10.1.2.1
                  - dest: 10.1.22.0/24
                    next_hops:
                      - forward_router_address: 10.1.2.1
        state: overridden    # config で指定したルート以外は削除される
```

このように「state: overridden」は設定の削除を伴います。挙動をよく確認したい場合は、ansible-

playbook コマンドのチェックモードを利用し、投入予定コマンドをあらかじめ確認することもできます。詳細は「6-6-4 投入予定のコマンドをチェックモードで確認する」を参照してください。

関　連

▷ cisco.ios.ios_static_routes モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_static_routes_module.html

6-4-2　OSPFv2 の設定をする

キーワード

▷ cisco.ios.ios_ospfv2 モジュール

▷ cisco.ios.ios_ospf_interfaces モジュール

▷ OSPFv2

方　法

OSPFv2 のプロセス単位の設定をするには cisco.ios.ios_ospfv2 モジュールを利用します。インターフェイスごとの設定をするには、cisco.ios.ios_ospf_interfaces モジュールを利用します。

OSPFv2 を有効化し、インターフェイスにコストを設定するプレイブックは List 6-25 のとおりです。

List 6-25　cisco.ios.ios_ospfv2、cisco.ios.ios_ospf_interfaces モジュールの利用例:
./router-switch/ios_ospfv2.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    # OSPFv2 の設定
    - name: set OSPFv2
      cisco.ios.ios_ospfv2:
        config:
          processes:
            - process_id: 1
              network:
                - address: 10.1.1.0
                  wildcard_bits: 0.0.0.255
```

```
15:                   area: 0
16:       state: merged
17:
18:     # インターフェイスコストの設定
19:     - name: set interface cost
20:       cisco.ios.ios_ospf_interfaces:
21:         config:
22:           - name: GigabitEthernet0/3
23:             address_family:
24:               - afi: ipv4
25:                 cost:
26:                   interface_cost: 100
27:         state: merged
```

解　説

OSPFv2 のプロセス単位の設定をする cisco.ios.ios_ospfv2 モジュールの主なパラメータは Table 6-17 のとおりです。

Table 6-17　cisco.ios.ios_ospfv2 モジュールの主なパラメータ

パラメータ名				説明
config				設定内容を指定する
	processes			プロセスをリストで指定する
		process_id		プロセス ID を指定する
		router_id		ルータ ID を指定する
		network		ルーティングを有効にするネットワークをリストで指定する
			address	アドレスを指定する
			wildcard_bits	ワイルドカードビットを指定する
			area	エリア ID を指定する
state				config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定した OSPF 設定をマージする

前掲のプレイブックの場合、Operation 6-19 のコマンドを投入します。

Operation 6-19　コマンド投入例

```
router ospf 1
network 10.1.1.0 0.0.0.255 area 0
```

インターフェイスごとに OSPF の設定をする cisco.ios.ios_ospf_interfaces モジュールの主なパラメータは Table 6-18 のとおりです。

Table 6-18　cisco.ios.ios_ospf_interfaces モジュールの主なパラメータ

パラメータ名				説明
config				設定内容をリストで指定する
	name			インターフェイス名を指定する
	address_family			Address Family をリストで指定する
		afi		Address Family Identifier を指定する（ipv4、ipv6）
		cost		コストを指定する
			interface_cost	インターフェイスコストを指定する
state				config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定したインターフェイスごとの OSPFv2 設定をマージする

前掲のプレイブックの場合、Operation 6-20 のコマンドを投入します。

Operation 6-20　コマンド投入例

```
interface GigabitEthernet0/3
ip ospf cost 100
```

関　連

▷ cisco.ios.ios_ospfv2 モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_ospfv2_module.html

▷ cisco.ios.ios_ospf_interfaces モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_ospf_interfaces_module.html

6-4-3　BGP の設定をする

キーワード

▷ cisco.ios.ios_bgp_global モジュール

▷ cisco.ios.ios_bgp_address_family モジュール

方　法

BGP 全体の設定するには cisco.ios.ios_bgp_global モジュールを利用します。BGP の Address Family レベルの設定をするには、cisco.ios.ios_bgp_address_family モジュールを利用します。

BGP を有効化し、IPv4 Unicast におけるネイバーの指定やルートマップの適用、広報するネットワークを設定するプレイブックは List 6-26 のとおりです。

List 6-26　cisco.ios.ios_bgp_global、cisco.ios.ios_bgp_address_family モジュールの利用例: ./router-switch/ios_bgp.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    # BGP グローバル設定
    - name: set BGP global
      cisco.ios.ios_bgp_global:
        config:
          as_number: 65001
          neighbor:
            - address: 172.16.1.2
              remote_as: 65002
            - address: 172.16.2.2
              remote_as: 65002
        state: merged

    # BGP ipv4 unicast 設定
    - name: set BGP for ipv4 unicast
      cisco.ios.ios_bgp_address_family:
        config:
          as_number: 65001
          address_family:
            - afi: ipv4
              neighbor:
                - address: 172.16.1.2
```

```
27:                   activate: true
28:                   route_maps:
29:                     - name: ROUTE-MAP01
30:                       in: true
31:                 - address: 172.16.2.2
32:                   activate: true
33:               network:
34:                 - address: 10.1.1.0
35:                   mask: 255.255.255.0
36:         state: merged
```

解　説

BGP 全体の設定をする cisco.ios.ios_bgp_global モジュールの主なパラメータは Table 6-19 のとおりです。

Table 6-19　cisco.ios.ios_bgp_global モジュールの主なパラメータ

パラメータ名			説明
config			設定内容を指定する
	as_number		自身の AS 番号を指定する
	neighbor		ネイバーをリストで指定する
		address	ネイバーのアドレスを指定する
		remote_as	ネイバーの AS 番号を指定する
state			config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定した設定をマージする

前掲のプレイブックの場合、Operation 6-21 のコマンドを投入します。

Operation 6-21　コマンド投入例

```
router bgp 65001
neighbor 172.16.1.2 remote-as 65002
neighbor 172.16.2.2 remote-as 65002
```

BGP の Address Family レベルの設定をする cisco.ios.ios_bgp_address_family モジュールの主なパラメータは Table 6-20 のとおりです。

Table 6-20　cisco.ios.ios_bgp_address_family モジュールの主なパラメータ

パラメータ名				説明
config				設定内容を指定する
	as_number			自身の AS 番号を指定する
	address_family			Address Family をリストで指定する
		afi		Address Family を指定する（ipv4、ipv6 など）
		safi		Address Family modifier を指定する（unicast、multicast など）、省略時は unicast 扱い
		neighbor		ネイバーをリストで指定する
			address	ネイバーのアドレスを指定する
			activate	ネイバーを有効にするかどうか指定する
			route_maps	適用するルートマップをリストで指定する
				name 適用するルートマップ名を指定する
				in インバウンド方向に適用するかどうか指定する
				out アウトバウンド方向に適用するかどうか指定する
		network		広報するネットワークをリストで指定する
			address	広報するネットワークアドレスを指定する
			mask	広報するネットワークアドレスに対するマスクを指定する
state				config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定した設定をマージする

前掲のプレイブックの場合、Operation 6-22 のコマンドを投入します。

Operation 6-22　コマンド投入例

```
router bgp 65001
address-family ipv4
neighbor 172.16.1.2 activate
neighbor 172.16.1.2 route-map ROUTE-MAP01 in
neighbor 172.16.2.2 activate
network 10.1.1.0 mask 255.255.255.0
```

応　用

BGP 全体の設定を削除、つまり「no router bgp <AS 番号>」コマンド相当の処理をする場合は、cisco.ios.ios_bgp_global モジュールの state パラメータで「purged」を指定します。

List 6-27　BGP 全体の設定を削除する例: ./router-switch/ios_bgp_advanced.yml

```
1: ---
2: - hosts: rt
3:   gather_facts: false
4:
5:   tasks:
6:     # BGP グローバル設定の削除
7:     - name: delete BGP global
8:       cisco.ios.ios_bgp_global:
9:         state: purged
```

なお、cisco.ios.ios_bgp_global モジュールの state パラメータとしては「deleted」も指定できますが、「deleted」は「router bgp <AS 番号>」より下の階層の設定が削除されます。「router bgp <AS 番号>」自体は残るという点で「purged」とは異なりますので、使い分けてください。

関　連

▷ cisco.ios.ios_bgp_global モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_bgp_global_module.html

▷ cisco.ios.ios_bgp_address_family モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_bgp_address_family_module.html

▷ Ansible 公式ブログによる「state: purged」の説明

https://www.ansible.com/blog/ansible-network-resource-purge-parameter

6-5　運用管理設定

Syslog や LLDP（Link Layer Discovery Protocol）などの運用管理に関する機能を設定するプレイブックを紹介します。運用管理機能は、多数の機器に統一的に同じ設定を投入することが多く、このような場合も Ansible が活用できます。

6-5-1　ログの送信先 Syslog サーバを設定する

キーワード

▷ cisco.ios.ios_logging_global モジュール

▷ Syslog

方　法

ログを送信する先の Syslog サーバを指定するには cisco.ios.ios_logging_global モジュールを利用します。

ログの送信先として Syslog サーバ「192.168.1.2」を指定するプレイブックは List 6-28 のとおりです。

List 6-28　cisco.ios.ios_logging_global モジュールの利用例: ./router-switch/ios_logging_global.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    - name: set syslog
      cisco.ios.ios_logging_global:
        config:
          hosts:
            - hostname: 192.168.1.2
        state: merged
```

解　説

cisco.ios.ios_logging_global モジュールの主なパラメータは Table 6-21 のとおりです。

Table 6-21　cisco.ios.ios_logging_global モジュールの主なパラメータ

パラメータ名			説明
config			設定内容を指定する
	hosts		Syslog サーバをリストで指定する
		hostname	送信先の Syslog サーバのホスト名や IP アドレスを指定する
state			config パラメータで指定した設定の扱い方を指定する。 デフォルトは merged であり、指定したログ設定をマージする

前掲のプレイブックの場合、Operation 6-23 のコマンドを投入します。

Operation 6-23　コマンド投入例

```
logging host 192.168.1.2
```

関　連

▷ cisco.ios.ios_logging_global モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_logging_global_module.html

6-5-2　参照先 NTP サーバを設定する

キーワード

▷ cisco.ios.ios_ntp モジュール

▷ NTP（Network Time Protocol）

方　法

ネットワーク機器の時刻を NTP サーバの時刻と同期するために、参照先 NTP サーバを指定するには cisco.ios.ios_ntp モジュールを利用します。

NTP サーバ 192.168.1.3 を参照先 NTP サーバとして設定するプレイブックは List 6-29 のとおりです。

List 6-29　cisco.ios.ios_ntp モジュールの利用例: ./router-switch/ios_ntp.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    # NTP サーバの設定
    - name: set ntp server
      cisco.ios.ios_ntp:
        server: 192.168.1.3
        source_int: Loopback0
        state: present
```

解　説

cisco.ios.ios_ntp モジュールの主なパラメータは Table 6-22 のとおりです。

Table 6-22　cisco.ios.ios_ntp モジュールの主なパラメータ

パラメータ名	説明
server	NTP サーバのホスト名または IP アドレスを指定する
source_int	NTP サーバとの通信に送信元として利用するインターフェイスを指定する
state	設定した状態（present）にするか、削除した状態（absent）にするか指定する。デフォルトは present

前掲のプレイブックの場合、Operation 6-24 のコマンドを投入します。

Operation 6-24　コマンド投入例

```
ntp server 192.168.1.3
ntp source Loopback0
```

応　用

state パラメータに「absent」を指定すると設定を削除できます。

関　連

▷ cisco.ios.ios_ntp モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_ntp_module.html

6-5-3　LLDP を設定する

キーワード

▷ cisco.ios.ios_lldp_global モジュール

▷ cisco.ios.ios_lldp_interfaces モジュール

▷ LLDP（Link Layer Discovery Protocol）

方　法

機器全体の LLDP を有効化または無効化、各種タイマー値の設定をするには cisco.ios.ios_lldp_global モジュールを利用します。インターフェイスごとの LLDP の設定をするには、cisco.ios.ios_lldp_interfaces モジュールを利用します。

機器の LLDP を有効化し、特定のインターフェイスのみ LLDP パケットの送受信を無効化するプレイブックは List 6-30 のとおりです。

List 6-30　cisco.ios.ios_lldp_global、cisco.ios.ios_lldp_interfaces モジュールの利用例: ./router-switch/ios_lldp.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    - name: lldp run
      cisco.ios.ios_lldp_global:
        config:
          enabled: true
          holdtime: 30
          reinit: 3
          timer: 20
        state: merged

    - name: unset lldp interface
      cisco.ios.ios_lldp_interfaces:
        config:
          - name: GigabitEthernet0/4
            receive: false
            transmit: false
        state: merged
```

解　説

機器全体の LLDP の設定をする cisco.ios.ios_lldp_global モジュールの主なパラメータは Table 6-23 のとおりです。

Table 6-23　cisco.ios.ios_lldp_global モジュールの主なパラメータ

パラメータ名		説明
config		設定内容を指定する
	enabled	LLDP の有効、無効を指定する
	holdtime	情報が廃棄されるまでの時間（ホールドタイム）を指定する（秒）
	reinit	初期化遅延時間を指定する（秒）
	timer	LLDP パケットが送信される間隔を指定する（秒）
state		config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定した設定をマージする

前掲のプレイブックの場合、Operation 6-25 のコマンドを投入します。

Operation 6-25　コマンド投入例

```
lldp holdtime 30
lldp run
lldp timer 20
lldp reinit 3
```

インターフェイスごとの LLDP の設定をする cisco.ios.ios_lldp_interfaces モジュールの主なパラメータは Table 6-24 のとおりです。

Table 6-24　cisco.ios.ios_lldp_interfaces モジュールの主なパラメータ

パラメータ名		説明
config		設定内容をリストで指定する
	name	設定対象のインターフェイス名を指定する
	receive	LLDP パケットの受信の有効、無効を指定する
	transmit	LLDP パケットの送信の有効、無効を指定する
state		config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定したインターフェイスの LLDP 設定をマージする

前掲のプレイブックの場合、Operation 6-26 のコマンドを投入します。

Operation 6-26　コマンド投入例

```
interface GigabitEthernet0/4
no lldp receive
no lldp transmit
```

応　用

機器全体の LLDP を無効化し、各種タイマー値をデフォルトに戻すには、cisco.ios.ios_lldp_global モジュールの state パラメータで「deleted」を指定します。

List 6-31　機器全体の LLDP の無効化とタイマー値の初期化: ./router-switch/ios_lldp_advanced.yml

```
1: ---
2: - hosts: rt
3:   gather_facts: false
4:
5:   tasks:
6:     - name: delete lldp global setting
```

```
7:      cisco.ios.ios_lldp_global:
8:        state: deleted
```

関　連

▷ cisco.ios.ios_lldp_global モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_lldp_global_module.html

▷ cisco.ios.ios_lldp_interfaces モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_lldp_interfaces_module.html

6-5-4　アクセスリストを設定する

キーワード

▷ cisco.ios.ios_acls モジュール

▷ cisco.ios.ios_acl_interfaces モジュール

▷ ACL (Access Control List)

方　法

アクセスリストを定義するには cisco.ios.ios_acls モジュールを利用します。定義したアクセスリストをインターフェイスに適用するには、cisco.ios.ios_acl_interfaces モジュールを利用します。

拡張アクセスリストを定義し、GigabitEthernet0/4 の in 方向に適用するプレイブックは List 6-32 のとおりです。

List 6-32　cisco.ios.ios_acls、cisco.ios.ios_acl_interfaces モジュールの利用例: ./router-switch/ios_acls.yml

```
1: ---
2: - hosts: rt
3:   gather_facts: false
4:
5:   tasks:
6:     # アクセスリストの定義
7:     - name: define ACL
8:       cisco.ios.ios_acls:
9:         config:
```

```
10:           - afi: ipv4
11:             acls:
12:               - name: 100
13:                 acl_type: extended
14:                 aces:
15:                   - sequence: 10
16:                     grant: permit
17:                     protocol: tcp
18:                     source:
19:                       address: 10.2.1.0
20:                       wildcard_bits: 0.0.0.255
21:                     destination:
22:                       address: 10.1.12.0
23:                       wildcard_bits: 0.0.0.255
24:                       port_protocol:
25:                         eq: 22
26:                   - sequence: 90
27:                     grant: deny
28:                     protocol: ip
29:                     source:
30:                       any: true
31:                     destination:
32:                       any: true
33:         state: merged
34:
35:     # アクセスリストの適用
36:     - name: apply ACL
37:       cisco.ios.ios_acl_interfaces:
38:         config:
39:           - name: GigabitEthernet0/4
40:             access_groups:
41:               - afi: ipv4
42:                 acls:
43:                   - name: 100
44:                     direction: in
45:         state: merged
```

解　説

アクセスリストを定義する cisco.ios.ios_acls モジュールの主なパラメータは Table 6-25 のとおりです。

Table 6-25　cisco.ios.ios_acls モジュールの主なパラメータ

パラメータ名						説明
config						設定内容をリストで指定する
	afi					Address Family Identifier を指定する（ipv4、ipv6）
	acls					アクセスリストの内容をリストで指定する
		name				番号か名前を指定する
		acl_type				アクセスリストのタイプを指定する（extended、standard）
		aces				アクセスリスト内のエントリをリストで指定する
			sequence			シーケンス番号を指定する
			grant			アクションを指定する（permit、deny）
			protocol			プロトコルを指定する
			source			送信元を指定する
				address		送信元アドレスを指定する
				wildcard_bits		送信元アドレスに対するワイルドカードマスクを指定する
				any		任意の送信元アドレスに一致させるかどうか
			destination			宛先を指定する
				address		宛先アドレスを指定する
				wildcard_bits		宛先アドレスに対するワイルドカードマスクを指定する
				any		任意の宛先アドレスに一致させるかどうか
				port_protocol		宛先ポート番号を指定する
					eq	一致するポート番号を指定する。ネットワーク機器側で、プロトコル名に解釈されるポート番号は、冪等性担保のためプロトコル名で指定する（例:80 ではなく www）
state						config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定したアクセスリストの定義設定をマージする

前掲のプレイブックの場合、Operation 6-27 のコマンドを投入します。

Operation 6-27　コマンド投入例

```
ip access-list extended 100
```

```
10 permit tcp 10.2.1.0 0.0.0.255 10.1.12.0 0.0.0.255 eq 22
90 deny ip any any
```

アクセスリストをインターフェイスに適用する cisco.ios.ios_acl_interfaces モジュールの主なパラメータは Table 6-26 のとおりです。

Table 6-26　cisco.ios.ios_acl_interfaces モジュールの主なパラメータ

パラメータ名				説明
config				設定内容をリストで指定する
	name			アクセスリストを適用するインターフェイス名を指定する
	access_groups			適用する access-group をリストで指定する
		afi		Address Family Identifier を指定する（ipv4、ipv6）
		acls		アクセスリスト適用設定をリストで指定する
			name	適用するアクセスリストの名前を指定する
			direction	アクセスリストを適用する方向を指定する（in、out）
state				config パラメータで指定した設定の扱い方を指定する。デフォルトは merged であり、指定したアクセスリストの適用設定をマージする

前掲のプレイブックの場合、Operation 6-28 のコマンドを投入します。

Operation 6-28　コマンド投入例

```
interface GigabitEthernet0/4
ip access-group 100 in
```

応　用

プレイブック内で指定したアクセスリストの定義のみを残して、他の定義を削除したい場合は、state パラメータで「overridden」を指定します。たとえば IPv4 のアクセスリスト「110」が設定されている状態で、前掲のプレイブックの cisco.ios.ios_acls モジュールの state パラメータを「overridden」に変更したプレイブックを実行すると、アクセスリスト「110」は削除され、アクセスリスト「100」のみになります。

補　足

cisco.ios コレクションのバージョン 2.0.1 までの cisco.ios.ios_acls モジュールには、source の wild card_bits パラメータで指定した値が、destination の wildcard_bits パラメータの値としても利用されてしまう不具合がありました[*17]。バージョン 2.1.0 で修正されたため、本章で扱うバージョン 2.3.0 では問題ありません。

関　連

▷ cisco.ios.ios_acls モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_acls_module.html

▷ cisco.ios.ios_acl_interfaces モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_acl_interfaces_module.html

6-5-5　バナーメッセージを設定する

キーワード

▷ cisco.ios.ios_banner モジュール

方　法

ログインプロンプト表示時やログイン後などのタイミングで表示する、バナーメッセージを設定するには cisco.ios.ios_banner モジュールを利用します。

ログインプロンプト表示時にバナーメッセージを表示するプレイブックは List 6-33 のとおりです。

List 6-33　cisco.ios.ios_banner モジュールの利用例: ./router-switch/ios_banner.yml

```
1: ---
2: - hosts: rt
3:   gather_facts: false
4:
5:   tasks:
6:     # ログインバナーの設定
```

＊17　関連 issue
https://github.com/ansible-collections/cisco.ios/issues/255

```
 7:    - name: set banner
 8:      cisco.ios.ios_banner:
 9:        banner: login
10:        text: |-
11:          --------- NOTICE ---------
12:            PRODUCTION environment
13:          --------------------------
14:        state: present
```

解　説

cisco.ios.ios_banner モジュールの主なパラメータは Table 6-27 のとおりです。

Table 6-27　cisco.ios.ios_banner モジュールの主なパラメータ

パラメータ名	説明
banner	バナーの種類を選択する。login、motd、exec などを指定できる
text	ターミナルに表示させたいテキストを指定する
state	設定した状態（present）にするか、削除した状態（absent）にするか指定する。デフォルトは present

前掲のプレイブックの場合、Operation 6-29 のコマンドを投入します。

Operation 6-29　コマンド投入例

```
banner login @
--------- NOTICE ---------
  PRODUCTION environment
--------------------------
@
```

これにより、ネットワーク機器に接続する際、設定したバナーが Operation 6-30 のように表示されます。

Operation 6-30　ログインバナー表示例

```
$ ssh admin@192.168.1.101

--------- NOTICE ---------
  PRODUCTION environment
--------------------------
Password:
```

応　用

text パラメータで、file や template ルックアッププラグインを利用すると、別ファイルで定義した内容を読み込んでバナーとして指定できます。

List 6-34　Jinja2 テンプレートによるバナー設定例: ./router-switch/ios_banner_advanced.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    # Jinja2 テンプレートを利用してログインバナーを指定
    - name: set banner
      cisco.ios.ios_banner:
        banner: login
        text: "{{ lookup('ansible.builtin.template', 'banner.j2') }}"
        state: present
```

関　連

▷ cisco.ios.ios_banner モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_banner_module.html

6-5-6　ユーザを追加する

キーワード

▷ cisco.ios.ios_user モジュール

方　法

ネットワーク機器にログインするためのユーザを追加するには、cisco.ios.ios_user モジュールを利用します。

ユーザ名を「ansible」、パスワードを「testpassword」、privilege を「15」に設定するプレイブックは List 6-35 のとおりです。

List 6-35　cisco.ios.ios_user モジュールの利用例: ./router-switch/ios_user.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    - name: add user
      cisco.ios.ios_user:
        name: ansible
        configured_password: testpassword
        privilege: 15
        password_type: secret
        state: present
```

解　説

cisco.ios.ios_user モジュールの主なパラメータは Table 6-28 のとおりです。

Table 6-28　cisco.ios.ios_user モジュールの主なパラメータ

パラメータ名	説明
name	追加するユーザ名を指定する
configured_password	平文のパスワードを指定する
privilege	特権レベルを指定する。1〜15 を指定できる
password_type	パスワードのタイプを指定する。secret と password を選択できる
update_password	パスワードを更新する条件を指定する。on_create ではユーザ作成時に、always では常に更新する。デフォルトは always
sshkey	公開鍵ファイルの内容を指定する。リストによって複数指定できる
state	存在する状態（present）にするか、削除した状態（absent）にするか指定する。デフォルトは present

前掲のプレイブックの場合、Operation 6-31 のコマンドを投入します。

Operation 6-31　コマンド投入例

```
username ansible privilege 15
username ansible secret testpassword
```

デフォルトではユーザが存在する場合でも プレイブック実行のたびにパスワードを更新します。その結果、タスクの実行ステータスが毎回「changed」になります。ユーザを追加するときのみパスワードを更新する場合は、update_password パラメータで「on_create」を指定します。

応　用

configured_password パラメータの代わりに、hashed_password パラメータを指定すると、ハッシュ化したパスワードを指定できます。hashed_password パラメータ配下の type パラメータには、5（MD5）、8（PBKDF2）、9（SCRYPT）などのハッシュアルゴリズムを指定します。ネットワーク機器側の対応状況を機器のマニュアルなどで確認の上、選択してください。

List 6-36　PBKDF2 でハッシュしたパスワードを指定する例: ./router-switch/ios_user_advanced.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    - name: specifies hashed password
      cisco.ios.ios_user:
        name: ansible
        hashed_password:
          type: 8    # PBKDF2
          # type に応じたハッシュ化文字列
          value: $8$xmOD2RT3ayddss$Alc4fB09KP1o9LCOyHJoMj/eP4fA3i1qJf8cDKSfkpk
        privilege: 15
```

関　連

▷ cisco.ios.ios_user モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_user_module.html

6-6　その他の設定

これまで紹介してきた設定変更のプレイブックには、その設定に特化した専用のモジュールを利用しました。しかし、実運用では専用モジュールがない、またはモジュールがあっても対応するパラメータがない設定を扱うこともあります。本節では、専用モジュールや対応パラメータがない範囲の設定をするためのプレイブックや、投入予定のコマンドをチェックモードで確認する方法を紹介します。

6-6-1　任意の設定コマンドを実行する

キーワード

▷ cisco.ios.ios_config モジュール

方　法

任意の設定コマンドを実行するには cisco.ios.ios_config モジュールを利用します。ここでは専用モジュールがない SNMP（Simple Network Management Protocol）コミュニティ名の設定を、cisco.ios.ios_config モジュールで行う例を紹介します。

List 6-37　cisco.ios.ios_config モジュールの利用例: ./router-switch/ios_config.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    # SNMP コミュニティ名の設定
    - name: set snmp community name
      cisco.ios.ios_config:
        lines:
          - "snmp-server community {{ item.community_name }} {{ item.privilege }}"
      loop:
        - community_name: public
          privilege: RO
        - community_name: secret
          privilege: RW
```

解　説

cisco.ios.ios_config モジュールの主なパラメータは Table 6-29 のとおりです。

Table 6-29　cisco.ios.ios_config モジュールの主なパラメータ

パラメータ名	説明
lines	実行するコマンドをリストで指定する
parents	モード移行するコマンドをリストで指定する
src	実行するコマンドの Jinja2 テンプレートまたはテキストファイル名を指定する。lines や parents パラメータとは併用不可
save_when	running-config を startup-config に保存する条件を指定する（always、never、modified、changed）。デフォルトは never であり、保存しない

前掲のプレイブックの場合、Operation 6-32 のコマンドを投入します。

Operation 6-32　コマンド投入例

```
snmp-server community public RO
snmp-server community secret RW
```

このように、手動と同じような感覚でコマンドを指定して投入できます。グローバルコンフィギュレーションモードに移行する「configure terminal」コマンドは、モジュール内部で暗黙的に実行されるため、プレイブック側では指定しません。また、変数や Jinja2 テンプレートを利用（「6-6-2 Jinja2 テンプレートで生成したコマンドを実行する」参照）することで効率的にコマンドを生成できます。

応　用

parents パラメータを利用すると、グローバルコンフィギュレーションモードから他のコンフィギュレーションモードに移行するためのコマンドを指定できます。

List 6-38 のプレイブックは、インターフェイス「GigabitEthernet0/3」に HSRP（Hot Standby Router Protocol）の設定をする例です。

List 6-38　parents パラメータの利用例: ./router-switch/ios_config_advanced.yml

```
1: ---
2: - hosts: rt
3:   gather_facts: false
4:
5:   tasks:
6:     # HSRP の設定
7:     - name: set HSRP
8:       cisco.ios.ios_config:
9:         lines:
10:          - standby version 2
11:          - standby 1 ip 10.1.1.254
12:          - standby 1 priority 110
13:        parents:
14:          - interface GigabitEthernet0/3
```

parents パラメータで指定した「interface GigabitEthernet0/3」を先に実行し、インターフェイスコンフィギュレーションモードに移行してから、lines パラメータで指定したコンフィグを実行します。

Operation 6-33　コマンド投入例

```
interface GigabitEthernet0/3
```

```
standby version 2
standby 1 ip 10.1.1.254
standby 1 priority 110
```

なお「interface GigabitEthernet0/3」のようなモード移行するためのコマンドは、lines パラメータで指定しないように注意してください。lines パラメータで「interface GigabitEthernet0/3」を指定してしまうと、現状のコンフィグに「interface GigabitEthernet0/3」がある場合「interface GigabitEthernet0/3」は実行しません。その結果、グローバルコンフィギュレーションのままインターフェイスコンフィギュレーション向けのコマンドを実行するため、意図どおり動作しません。そのため上記の例のように、モード移行するためのコマンドは parents パラメータに指定してください。

補　足

cisco.ios.ios_config モジュールは、タスクの実行ステータスが「changed」の場合に冪等性担保に関する警告が表示されます。

Operation 6-34　冪等性担保に関する警告

```
To ensure idempotency and correct diff the input configuration lines should be
similar to how they appear if present in the running configuration on device in
cluding the indentation
```

冪等性担保のためには、モジュールで指定するコマンドは「show running-config」で表示される形式に合わせることが重要です。公式ドキュメントの FAQ[*18] としても掲載されています。

関　連

▷ cisco.ios.ios_config モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_config_module.html

▷ 6-6-2 Jinja2 テンプレートで生成したコマンドを実行する

6-6-2　Jinja2 テンプレートで生成したコマンドを実行する

キーワード

▷ cisco.ios.ios_config モジュール

＊ 18　Why do the *_config modules always return changed=true with abbreviated commands?
https://docs.ansible.com/ansible/latest/network/user_guide/faq.html#why-do-the-config-modules-always-return-changed-true-with-abbreviated-commands

▷ Jinja2

方 法

Jinja2 テンプレートを利用してコマンドを生成して実行するには、cisco.ios.ios_config モジュールの src パラメータを利用します。Jinja2 の for 文や if 文などのさまざまな制御文やフィルタを利用して、柔軟にコンフィグを生成できます。

ここでは 2 つのインターフェイスに対して HSRP の設定をする例を紹介します。

List 6-39 cisco.ios.ios_config の src パラメータ利用例: ./router-switch/ios_config_src.yml

```
---
- hosts: rt
  gather_facts: false

  vars:
    # Jinja2 テンプレート内で利用する変数の定義
    hsrp_config:
      - interface: GigabitEthernet0/3
        version: 2
        group: 1
        vip: 10.1.1.254
        priority: 110
      - interface: GigabitEthernet0/4
        version: 2
        group: 1
        vip: 172.31.1.254
        priority: 110

  tasks:
    # HSRP の設定
    - name: set HSRP by Jinja2
      cisco.ios.ios_config:
        src: hsrp.j2
```

src パラメータで指定する Jinja2 テンプレートは List 6-40 のとおりです。

List 6-40 HSRP 設定生成用 Jinja2 テンプレート: ./router-switch/templates/hsrp.j2

```
{% for c in hsrp_config %}
interface {{ c.interface }}
 standby version {{ c.version }}
 standby {{ c.group }} ip {{ c.vip }}
```

```
5:  standby {{ c.group }} priority {{ c.priority }}
6: {% endfor %}
```

解　説

cisco.ios.ios_config モジュールの src パラメータで、Jinja2 テンプレートのファイル名を指定します。

src パラメータは parents パラメータと併用できませんが、テンプレートファイル内でモード移行するコマンドを指定できます。その際、モード移行後のコマンドは、running-config の表示形式と同じようにインデントを入れて階層化する必要があります。「List 6-40 HSRP 設定生成用 Jinja2 テンプレート」でも standby コマンドにはスペースによるインデントを入れています。正しくインデントを入れないと running-config と正しく比較できず、冪等性を担保できない、コマンド実行エラーになるといった問題が発生してしまうので注意してください。

前掲のプレイブックと Jinja2 テンプレートの場合、Operation 6-35 のコマンドを投入します。変数「hsrp_config」内で定義したインターフェイスの分を繰り返してコマンドが生成され、実行されます。

Operation 6-35　コマンド投入例

```
interface GigabitEthernet0/3
standby version 2
standby 1 ip 10.1.1.254
standby 1 priority 110
interface GigabitEthernet0/4
standby version 2
standby 1 ip 172.31.1.254
standby 1 priority 110
```

関　連

▷ cisco.ios.ios_config モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_config_module.html

▷ Templating (Jinja2)

https://docs.ansible.com/ansible/latest/user_guide/playbooks_templating.html

▷ Jinja2 公式ドキュメント

https://jinja.palletsprojects.com/

▷ 6-6-1 任意の設定コマンドを実行する

6-6-3 コンフィグを保存する

キーワード

▷ cisco.ios.ios_config モジュール

方 法

設定変更モジュールは、あくまで running-config に対する設定変更のみを行います。startup-config への保存は、cisco.ios.ios_config モジュールの save_when パラメータを利用します。

cisco.ios.ios_interfaces モジュールで設定変更を行い、running-config と startup-config の間に差分が発生した場合に startup-config へ保存するプレイブックは List 6-41 のとおりです。

List 6-41 cisco.ios.ios_config モジュールの save_when パラメータの利用例:
./router-switch/ios_copy_run_start.yml

```
---
- hosts: rt
  gather_facts: false

  tasks:
    # description の変更
    - name: set interface description
      cisco.ios.ios_interfaces:
        config:
          - name: GigabitEthernet0/3
            description: changed_description
            enabled: true

    # running-config と startup-config に差分がある場合に保存
    - name: save config
      cisco.ios.ios_config:
        save_when: modified
```

解 説

cisco.ios.ios_config モジュールの save_when パラメータでは、running-config を startup-config に保存する条件を指定します。指定できる条件は Table 6-30 のとおりです。

Table 6-30　save_when パラメータで指定する条件

条件	説明
always	常に保存する
never	保存しない（デフォルト）
changed	自身のタスクによってコンフィグに変更が発生した場合に保存する
modified	running-config と startup-config に差分がある場合に保存する

modified と changed の違いについて補足します。cisco.ios.ios_config モジュール自身も lines や src パラメータによって任意の設定変更コマンドを指定でき、save_when パラメータとも併用できます。cisco.ios.ios_config モジュールによる設定変更が発生した場合に（タスクの実行ステータスが changed）startup-config に保存するのが「changed」です。

一方、modified は cisco.ios.ios_config モジュール自身による設定変更かどうかにかかわらず、running-config を startup-config に差分があると保存します。

関　連

▷ cisco.ios.ios_config モジュール

https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_config_module.html

6-6-4　投入予定のコマンドをチェックモードで確認する

キーワード

▷ ansible-playbook --check オプション

▷ ansible-playbook -v オプション

方　法

設定変更モジュールは、プレイブックで指定された各種パラメータの内容に応じてコマンドを生成して投入します。ansible-playbook コマンドの--check と-v オプションを利用することで、実際には設定変更をせずに投入予定のコマンドを確認できます。

「6-4-1 スタティックルートを設定する」で紹介したプレイブックを例にして、--check と-vvv オプション*19によって実行予定コマンドの確認する例を紹介します。

＊ 19　verbose モード有効化のオプションです。「v」の指定数で詳細度を調整できます。実際は「-v」でも戻り値は表示されますが、見やすさのため「-vvv」を紹介しています。

Operation 6-36　投入予定コマンドの確認例

```
$ ansible-playbook \
    -i ./router-switch/inventory.ini ./router-switch/ios_static_routes.yml \
    --check -vvv
...(略)...
    "commands": [
        "ip route 10.1.21.0 255.255.255.0 10.1.2.1",
        "ip route 10.1.22.0 255.255.255.0 10.1.2.1"
    ],
...(略)...
```

この結果により「ip route 10.1.21.0 255.255.255.0 10.1.2.1」と「ip route 10.1.22.0 255.255.255.0 10.1.2.1」が投入予定であることが分かります。チェックモードによるプレイブック実行のため、実際には設定変更されません。

解　説

公式ドキュメントの各モジュールの説明ページの「Return Values」にはモジュールの戻り値が掲載されています。cisco.ios コレクションの設定変更モジュールの場合は戻り値に「commands」が含まれます。「commands」には投入するコマンドがリスト形式で格納されます。チェックモードによるプレイブック実行でも同様です。

チェックモードに加えて「-vvv」オプションを併用してモジュールの戻り値を表示することで、設定変更をすることなく投入予定のコマンドをあらかじめ確認できます。

関　連

▷ Validating tasks: check mode and diff mode

https://docs.ansible.com/ansible/latest/user_guide/playbooks_checkmode.html

第７章
ロードバランサ

ロードバランサの設定では、多くの場合 GUI が利用されています。GUI による設定は、直観的で操作が容易であるというメリットがある半面、作業時間が長くなり、繰り返し行う作業ではミスが起こりやすいといった課題もあります。また、ロードバランサの運用では、SSL 証明書の更新などのような定期的に行う作業がありますが、こうした作業を Ansible で自動化すれば、運用工数は大幅に削減されます。

本章では F5 BIG-IP Local Traffic Manager（LTM）を対象に、運用時の課題を解決するプレイブックをいくつか紹介します。

7-1　ロードバランサの対応概要

各プレイブックの紹介に先立ち、Ansible がロードバランサに対してできることや環境の準備方法、各プレイブックの前提となる構成を説明します。

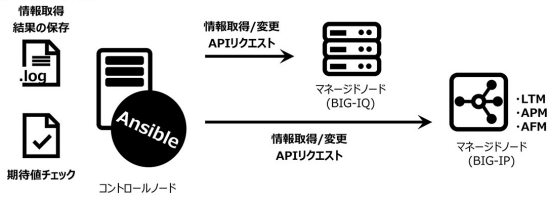
7-1-1　対応範囲

Ansible が対応するプラットフォームや、自動化できる内容を説明します。

■ 自動化できること

BIG-IP に対して、設定変更や情報取得を Ansible で実現できます。本章で紹介する LTM 以外にも、APM（Access Policy Manager）や AFM（Advanced Firewall Manager）などの設定変更や情報取得も可能です。BIG-IP を統合管理する BIG-IQ も Ansible で自動化できます（Figure 7-1）。

Figure 7-1　自動化できること



7-1-2　環境の準備

■ コントロールノード側の準備

コントロールノード側では、f5networks.f5_modules コレクションのインストールと環境変数の設定、そして Python インタープリタの設定という 3 つの作業を実施します。

(1) f5networks.f5_modules コレクションのインストール

本章では pip で ansible 4.2.0 をインストール時に同梱されている f5networks.f5_modules コレクションのバージョン 1.10.1 を利用します。

同梱バージョン以外を使いたい場合は、f5networks.f5_modules コレクションを「`ansible-galaxy collection install`」コマンドでインストールします。

Operation 7-1　f5networks.f5_modules コレクションのインストールコマンド実行例

```
$ ansible-galaxy collection install f5networks.f5_modules
```

(2) 環境変数の設定

続いて、BIG-IP にログインするための認証情報を環境変数として設定します。環境変数を利用することでモジュール利用時にユーザ名やパスワードなどの指定が不要になり、プレイブックの可読性が向上します。

Operation 7-2　環境変数の設定実行例

```
$ export F5_SERVER='IPaddress'
$ export F5_USER='username'
$ export F5_PASSWORD='password'
$ export F5_SERVER_PORT=443
$ export F5_VALIDATE_CERTS='False'
```

環境変数を設定した場合のプレイブック（List 7-1 ）と設定をしないプレイブック（List 7-2 ）を比較します。記述量が減り、処理内容が分かりやすくなります。

List 7-1　環境変数利用時のプレイブック

```
1: ---
2: - hosts: bigip
3:   connection: ansible.builtin.local
4:   gather_facts: false
5:
6:   tasks:
7:     - name: fetch bigip_device_info
8:       f5networks.f5_modules.bigip_device_info:
9:         gather_subset:
10:          - system-info
```

List 7-2　環境変数を利用しないプレイブック

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false

  tasks:
    - name: fetch bigip_device_info
      f5networks.f5_modules.bigip_device_info:
        gather_subset:
          - system-info
        provider:
          server: lb.mydomain.com
          user: admin
          password: secret
```

(3) Python インタープリタの設定

venv などを利用して Ansible を仮想環境にインストールしている場合は、モジュール実行時に利用する Python インタープリタの設定も必要です。

設定には `ansible_python_interpreter` 変数か、`ansible.cfg` の defaults セクションの「`interpreter_python`」を利用します。たとえば「`ansible`」を「`/home/ansible/myvenv`」という venv にインストールした場合は「`/home/ansible/myvenv/bin/python`」を指定します（List 7-3 ）。

ほかには、venv のパスを直接指定するのではなく、マジック変数「`ansible_playbook_python`」を指定する方法もあります。

List 7-3　ansible.cfg で Python インタープリタを設定する例

```
[defaults]
interpreter_python =/home/ansible/myvenv/bin/python
```

■ マネージドノード（BIG-IP）側の準備

マネージドノードである BIG-IP には Ansible を利用するための設定はとくに必要ありません。BIG-IP のソフトウェアバージョンは「`12`」以上である必要があります。

7-1-3 f5networks.f5_modules コレクションのモジュールの基本仕様

プレイブックに記載されたパラメータから iControlREST の URL とペイロードを生成します。モジュール内で指定された state パラメータに対応する HTTP メソッドでリクエストを送信します。

■ モジュール一覧と分類

f5networks.f5_modules コレクションには、BIG-IP のモジュールと BIG-IQ（BIG-IP を集中管理するための製品）モジュールの 2 種類のモジュールが存在します。それぞれ「bigip_*」、「bigiq_*」という命名ルールで作成されています。BIG-IP 用のモジュールはさらに細かく機能ごとに分かれています。たとえば、BIG-IP APM 用（BIG-IP Access Policy Manager）のモジュールであれば「bigip_apm_*」といった命名ルールです。しかし、LTM に関しては「bigip_*」の命名ルールで作成されており、必ずしも統一されているわけではないので、モジュールを探す際には注意してください。

■ 冪等性

f5networks.f5_modules コレクションのモジュールは、基本的には冪等性が担保されています。しかし一部モジュールには、注意すべき点もあります。たとえば、f5networks.f5_modules.bigip_pool_member モジュールで aggregate パラメータを有効にする場合です。このパラメータではノードの名前と IP アドレスを入力する必要がありますが、冪等性のチェックが行われるのはノードの名前のみです。このように冪等性が担保されているとはいえ、モジュールによっては仕組みが異なります。

■ 接続時に利用する特別な変数

BIG-IP への接続時にはコネクションプラグインを ansible.buitin.local、もしくはタスクに「delegate_to: localhost」を指定して実行する必要があります。プレイで定義するタスクが BIG-IP への作業のみの場合はコネクションプラグインを ansible.buitin.local に指定します。

Table 7-1 接続に利用する特別な変数

変数名	説明
ansible_connection	利用するコネクションプラグイン。 BIG-IP では「ansible.buitin.local」を利用する

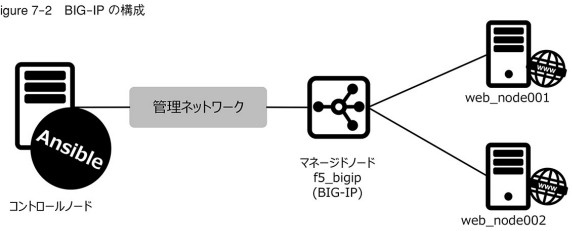
7-1-4　本章の前提構成

7-2 節以降で掲載する各プレイブックの前提となる環境や、インベントリ、変数定義ファイルについて説明します。

■ 環境

本章で利用する BIG-IP の Version は 16.0.1.1 を利用します。AWS 上に構築し、2 台のサーバが配下に設置されています（Figure 7-2）。

Figure 7-2　BIG-IP の構成



■ インベントリと変数定義ファイル

F5 BIG-IP モジュールでは「`connection: ansible.buitin.local`」を利用するため、接続先は localhost になります。また、認証情報は環境変数で設定しているため、ホスト変数ファイルやグループ変数ファイルは用意せず、List 7-4 のインベントリのみを用意します。

List 7-4　インベントリファイル: ./lb/inventory.ini

```
1: [bigip]
2: f5_bigip
```

本章の環境では、1 台の BIG-IP に対してプレイブックを実行するため、環境変数によって認証情報を設定します。環境変数やタスク内で認証情報を設定する場合、インベントリを作成せずにローカルホスト向けに実行するプレイブックを作成してもプレイブックの動作に差はありません。BIG-IP が複

数存在する環境では、認証情報を環境変数で設定せずにホスト変数やグループ変数に認証情報を定義することで作業対象を分けることもできます。

Column　2つのコレクション

BIG-IP を Ansible で操作するコレクションは 2 つ存在します。本章で紹介するのは f5networks.f5_modules コレクションですが、f5networks.f5_bigip コレクションと呼ばれる新たなコレクションも開発されています。特徴としては自動化処理が手続き型で行われるか宣言型で行われるかの違いです。

(1) f5networks.f5_modules コレクション

REST API を活用（手続き型）した F5 BIG-IP/BIG-IQ を管理できるコレクションです。v1 コレクションと呼びます。

(2) f5networks.f5_bigip コレクション*1

AS3（Application Services 3）、DO（Declarative Onboarding）、TS（Telemetry Streaming）、CFE（Cloud Failover Extension）などを活用（宣言型）した F5 BIG-IP/BIG-IQ を管理できるコレクションです。v2 コレクションと呼びます。

7-1-5　接続確認をする

環境変数などに設定した認証情報を確認するため、BIG-IP への接続確認を行います（List 7-5 ）。

BIG-IP への接続確認には f5networks.f5_modules.bigip_device_info モジュール*2を使うことを推奨します。f5networks.f5_modules.bigip_device_info モジュールは BIG-IP の情報をファクトとして収集するモジュールです。

＊1　2021/7/31 時点では Preview

＊2　BIG-IP の情報を収集する（f5networks.f5_modules.bigip_device_info モジュール）
https://docs.ansible.com/ansible/latest/collections/f5networks/f5_modules/bigip_device_info_module.html

List 7-5　接続確認用プレイブック: ./lb/bigip_device_info.yml

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false

  tasks:
    - name: fetch bigip_device_info
      f5networks.f5_modules.bigip_device_info:
        gather_subset:
          - system-info
```

f5networks.f5_modules.bigip_device_info モジュールの主なパラメータは Table 7-2 のとおりです。

Table 7-2　f5networks.f5_modules.bigip_device_info モジュールの主なパラメータ

パラメータ名	説明
gather_subset	収集するファクトのサブセットをリストで指定する（all、monitors、profiles など）。 この指定により収集したファクトは「ansible_facts」という変数名に格納される。 「!monitors」のように最初に「!」を付けると除外できる

f5networks.f5_modules.bigip_device_info モジュールの gather_subnet パラメータをリストで指定します。指定なしの場合はすべて（all）の情報取得になるため、処理に時間がかかります。取得したい情報が分かっている場合は、サブセットを指定し特定の情報のみを取得することを推奨します。指定できるサブセットは 75 個あり非常に多いため、一覧については公式ドキュメントを参照ください。本章で紹介するプレイブックに関連するサブセットは以下のとおりです。

- all
- client-ssl-profiles
- interfaces
- nodes
- ssl-keys
- monitors
- http-monitors
- irules
- self-ips
- virtual-servers
- profiles
- http-profiles
- ltm-pools
- ssl-certs
- vlans

7-2　ネットワーク設定

BIG-IP のネットワーク設定における、VLAN やセルフ IP などの基本的な設定内容について、Ansible で実施する方法を紹介します。設計によっては頻繁に設定追加が発生するため、プレイブックで実現

できると作業時間の短縮や作業品質の均一化を図るといったメリットがあります。

7-2-1　VLAN を作成する

キーワード

▷ f5networks.f5_modules.bigip_vlan モジュール

方　法

VLAN を作成するプレイブックを紹介します（List 7-6 ）。VLAN の作成時にはアサインするインターフェイスも合わせて指定する必要があります。本環境では、VLAN アサインは untagged インターフェイス 1.1 に対して実施します。

List 7-6　VLAN を作成するプレイブック: ./lb/bigip_vlan.yml

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false
  vars:
    assign_interface: 1.1
    tagging: untagged
    vlan_id: 11
    vlan_name: internal

  tasks:
    - name: create internal vlan
      f5networks.f5_modules.bigip_vlan:
        interfaces:
          - interface: "{{ assign_interface }}"
            tagging: "{{ tagging }}"
        name: "{{ vlan_name }}"
        tag: "{{ vlan_id }}"
        state: present
```

解　説

VLAN の作成には、f5networks.f5_modules.bigip_vlan モジュールを利用します。f5networks.f5_modules.bigip_vlan モジュールの主なパラメータは Table 7-3 のとおりです。

Table 7-3　f5networks.f5_modules.bigip_vlan モジュールの主なパラメータ

パラメータ名		説明
interfaces		VLAN を追加するインターフェイスのパラメータをリストで定義
	interface	インターフェイス名を定義
	tagging	tagged もしくは untagged を指定
name		VLAN 名を定義
tag		VLAN ID を定義

サンプルで利用している interfaces パラメータは、tagged_interfaces、または untagged_interfaces でも指定できます。パラメータを変えずに値だけを変えることで tagged インターフェイス、untagged インターフェイスを切り替えることができるため、interfaces のパラメータを利用することを推奨します。

関　連

▷ VLAN を管理する（f5networks.f5_modules.bigip_vlan モジュール）
`https://docs.ansible.com/ansible/latest/collections/f5networks/f5_modules/bigip_vlan_module.html`

7-2-2　セルフ IP を作成する

キーワード

▷ f5networks.f5_modules.bigip_selfip モジュール

方　法

セルフ IP の作成には f5networks.f5_modules.bigip_selfip モジュールを利用します（List 7-7）。

List 7-7　セルフ IP を作成するプレイブック: ./lb/bigip_self_ip.yml

```
1: ---
2: - hosts: bigip
```

```
 3:  connection: ansible.builtin.local
 4:  gather_facts: false
 5:
 6:  tasks:
 7:    - name: create selfip(non-flooting)
 8:      f5networks.f5_modules.bigip_selfip:
 9:        state: present
10:        name: "internal_selfip"
11:        address: "{{ '10.100.0.143/26' | ansible.netcommon.ipaddr('address') }}"
12:        netmask: "{{ '10.100.0.143/26' | ansible.netcommon.ipaddr('netmask') }}"
13:        vlan: "internal"
14:        allow_service: default
15:        traffic_group: /Common/traffic-group-local-only
16:
```

解　説

セルフ IP の作成には、f5networks.f5_modules.bigip_selfip モジュールを利用します。address パラメータと netmask パラメータは、それぞれの書式に合う値を抽出するために、ansible.netcommon.ipaddr フィルタを利用します。ansible.netcommon.ipaddr フィルタに必要な Python ライブラリの netaddr は、あらかじめ pip でインストールしておく必要があります。f5networks.f5_modules.bigip_selfip モジュールの主なパラメータは Table 7-4 のとおりです。

Table 7-4　f5networks.f5_modules.bigip_selfip モジュールの主なパラメータ

パラメータ名	説明
name	セルフ IP 名を定義
address	IP アドレスを定義
netmask	ネットマスクを定義
vlan	VLAN 名を定義
allow_service	ポートロックダウン設定の定義
traffic_group	トラフィックグループを定義（デフォルト: /Common/traffic-group-local-only）

ポートロックダウン設定の定義をする allow_service パラメータは、リストで定義できます。defalut ではなく、特定のプロトコル、ポート番号を指定する場合は以下のように定義します。

Operation 7-3　ポートロックダウン設定をカスタマイズする定義方法

```
...（ 略 ）...
       allow_service:
```

```
          - icmp
          - tcp:80
...（略）...
```

TCP、UDP のポート番号まで指定する場合は「tcp:**ポート番号**」、「udp:**ポート番号**」と定義します。ポート番号は省略し、プロトコル名のみ指定するとすべて許可（ALL）となります。また、共用 IP（Floating IP）を作成したい場合は、traffic_group パラメータを変更します。

関　連

▷ 2-3-5 IP アドレスを扱う

▷ セルフ IP を管理する（f5networks.f5_modules.bigip_selfip モジュール）

https://docs.ansible.com/ansible/latest/collections/f5networks/f5_modules/bigip_selfip_module.html

7-3　LTM 設定

本節では、Ansible による BIG-IP LTM の設定方法について解説します。ノードの作成やバーチャルサーバの設定など、普段の運用で実施する作業内容に焦点を当てて紹介します。

7-3-1　モニタを作成する

キーワード

▷ f5networks.f5_modules.bigip_monitor_http モジュール

方　法

HTTP モニタを作成するプレイブックを紹介します（List 7-8 ）。

List 7-8　HTTP モニタを作成するプレイブック: ./lb/bigip_http_monitor.yml

```
1: ---
2: - hosts: bigip
3:   connection: ansible.builtin.local
4:   gather_facts: false
5:
6:   tasks:
```

```
 7:     - name: create http monitor
 8:       f5networks.f5_modules.bigip_monitor_http:
 9:         state: present
10:         name: http_monitor
11:         ip: '*'
12:         port: '*'
13:         send: 'GET /\r\n'
14:         receive: ''
15:         interval: 5
```

解　説

モニタの作成は、f5networks.f5_modules.bigip_monitor_* モジュールを利用します。本サンプルでは HTTP モニタを作成するため、f5networks.f5_modules.bigip_monitor_http モジュールを利用します。

Table 7-5　f5networks.f5_modules.bigip_monitor_http モジュールの主なパラメータ

パラメータ名	説明
name	モニタ名を定義
ip	モニタ対象の IP を定義（デフォルト: *）
port	モニタ対象のポート番号を定義（デフォルト: *）
send	送信文字列を定義（デフォルト: GET /\r\n ）
receive	受信文字列を定義
interval	モニタの実行間隔（デフォルト: 5）

f5networks.f5_modules.bigip_monitor_http モジュール利用時の注意点は、ip パラメータ、port パラメータの指定方法と send パラメータ、receive パラメータの定義方法です。ip パラメータ、port パラメータはデフォルトでは「*」です。「*」はすべてのアドレスとすべてのポートが対象になります。send パラメータ、receive パラメータは文字列を直接定義します。ダブルクォーテーションで文字列を囲む際はバックスラッシュ（\）はエスケープ文字になるため扱いには注意が必要です。

モニタは、HTTP モニタ以外に 15 種類あります、モニタ系モジュールは以下のとおりです。必要に応じて利用してください。

- bigip_monitor_dns
- bigip_monitor_external
- bigip_monitor_ftp
- bigip_monitor_gateway_icmp
- bigip_monitor_http
- bigip_monitor_https
- bigip_monitor_icmp
- bigip_monitor_ldap
- bigip_monitor_mysql
- bigip_monitor_oracle
- bigip_monitor_smtp
- bigip_monitor_snmp_dca
- bigip_monitor_tcp
- bigip_monitor_tcp_echo
- bigip_monitor_tcp_half_open
- bigip_monitor_udp

関　連

▷ HTTP モニタを管理する（f5networks.f5_modules.bigip_monitor_http モジュール）
https://docs.ansible.com/ansible/latest/collections/f5networks/f5_modules/bigip_monitor_http_module.html

7-3-2　プロファイルを作成する

キーワード

▷ f5networks.f5_modules.bigip_profile_http モジュール

方　法

HTTP プロファイルを作成するには、f5networks.f5_modules.bigip_profile_http モジュールを利用します（List 7-9 ）。

List 7-9　プロファイルを作成するプレイブック: ./lb/bigip_http_profile.yml

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false

  tasks:
    - name: create http profile
      f5networks.f5_modules.bigip_profile_http:
        state: present
        name: http_profile
        insert_xforwarded_for: true

```

解　説

プロファイルの作成は、f5networks.f5_modules.bigip_profile_* モジュールを利用します。本サンプルでは HTTP プロファイルの作成に、f5networks.f5_modules.bigip_profile_http モジュールを利用します。

Table 7-6　f5networks.f5_modules.bigip_profile_http モジュールの主なパラメータ

パラメータ名	説明
name	プロファイル名を定義
insert_xforwarded_for	X-Forwarded-For ヘッダを有効化・無効化を指定

プロファイルは、HTTP プロファイル以外に 15 種類あります。プロファイル系モジュールは以下のとおりです、必要に応じて利用してください。

- bigip_profile_analytics
- bigip_profile_client_ssl
- bigip_profile_dns
- bigip_profile_fastl4
- bigip_profile_ftp
- bigip_profile_http
- bigip_profile_http2
- bigip_profile_http_compression
- bigip_profile_oneconnect
- bigip_profile_persistence_cookie
- bigip_profile_persistence_src_addr
- bigip_profile_udp
- bigip_profile_server_ssl
- bigip_profile_sip
- bigip_profile_tcp
- bigip_profile_persistence_universal

関　連

▷ HTTP プロファイルを管理する（f5networks.f5_modules.bigip_profile_http モジュール）

https://docs.ansible.com/ansible/latest/collections/f5networks/f5_modules/bigip_profile_http_module.html

7-3-3　ノードを作成する

キーワード

▷ f5networks.f5_modules.bigip_node モジュール

方　法

ノードの作成は、f5networks.f5_modules.bigip_node モジュールを利用します（List 7-10 ）。

List 7-10　ノードを作成するプレイブック: ./lb/bigip_node.yml

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false

  tasks:
    - name: create node
      f5networks.f5_modules.bigip_node:
        state: present
        name: web_node001
        address: 10.100.0.135
        monitors:
```

```
13:           - /Common/icmp
14:         availability_requirements:
15:           type: all
16:         connection_limit: 0
```

解　説

f5networks.f5_modules.bigip_node モジュールの主なパラメータは Table 7-7 のとおりです。

Table 7-7　f5networks.f5_modules.bigip_node モジュールの主なパラメータ

パラメータ名		説明
name		ノード名を定義
address		ノードの IP アドレスを定義
monitors		モニタを定義
availability_requirements		複数のモニタを定義する場合に利用
	type	モニタ監視のルールタイプを指定
	at_least	監視成功になるモニタ数を定義
connection_limit		コネクションリミット数を定義

monitors パラメータはリストで定義できます。複数のモニタを設定する場合は、availability_requirements パラメータを利用します。availability_requirements パラメータの type パラメータの値に at_least を指定した場合は、at_least パラメータで監視成功となるモニタ数を定義します。

f5networks.f5_modules.bigip_node モジュールはノードの状態を定義するため、state パラメータは作成（present）、削除（absent）以外も存在します。利用できる state パラメータは Table 7-8 のとおりです。

Table 7-8　f5networks.f5_modules.bigip_node モジュールで利用できる state パラメータ

パラメータ値	説明
present	ノードを作成する
absent	ノードを削除する
enabled	ノードを有効化する
disabled	ノードを無効化する
offline	ノードをオフラインにする

応　用

ノードのコネクションリミットを変更するプレイブックを紹介します（List 7-11 ）。

List 7-11　コネクションリミットを変更するプレイブック: ./lb/bigip_node_change_connection_limit.yml

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false

  tasks:
    - name: change connection limit
      f5networks.f5_modules.bigip_node:
        state: present
        name: web_node001
        address: 10.100.0.135
        monitors:
          - /Common/icmp
        availability_requirements:
          type: all
        connection_limit: 100 #コネクションリミットの変更
```

connection_limit パラメータの値を変更することで、コネクションリミットを変更できます。「0」を指定した場合リミット制限なしの設定になります。

関　連

▷ ノードを管理する（f5networks.f5_modules.bigip_node モジュール）

https://docs.ansible.com/ansible/latest/collections/f5networks/f5_modules/bigip_node_module.html

7-3-4　プールを作成する

キーワード

▷ f5networks.f5_modules.bigip_pool モジュール

▷ f5networks.f5_modules.bigip_pool_member モジュール

方　法

プールの作成とプールメンバの設定をするプレイブックを紹介します（List 7-12 ）。

List 7-12　プールを作成するプレイブック: ./lb/bigip_pool.yml

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false
  vars:
    pool_name: http_pool

  tasks:
    - name: create pool
      f5networks.f5_modules.bigip_pool:
        state: present
        name: "{{ pool_name }}"
        monitors: http_monitor
        lb_method: least-connections-member

    - name: add a pool member
      f5networks.f5_modules.bigip_pool_member:
        state: present
        pool: "{{ pool_name }}"
        aggregate:
          - name: web_node001
            address: 10.100.0.135
            port: 80
```

解　説

(1) プールの作成

プールの作成は、f5networks.f5_modules.bigip_pool モジュールを利用します。f5networks.f5_modules.bigip_pool モジュールの主なパラメータは Table 7-9 のとおりです。

Table 7-9　f5networks.f5_modules.bigip_pool モジュールの主なパラメータ

パラメータ名	説明
name	プール名を定義
monitors	モニタ名を定義
lb_method	振り分け方式を指定（デフォルト: round-robin）

(2) プールメンバの登録

プール作成後にメンバを登録するには、f5networks.f5_modules.bigip_pool_member モジュールを利用

します。f5networks.f5_modules.bigip_pool_member モジュールの主なパラメータは Table 7-10 のとおりです。

Table 7-10　f5networks.f5_modules.bigip_pool_member モジュールの主なパラメータ

パラメータ名		説明
pool		プール名を定義
aggregate		プールメンバ名をリストで定義
	name	ノード名を定義
	address	IP アドレスを定義
	port	ポート番号を定義

aggregate パラメータを利用することで、複数のノードを一度に追加できます。aggregate パラメータで定義している内容に誤りがあっても途中まで処理が実行され、タスクは失敗します。そのため、実行前に check_mode で設定に誤りがないか確認するなど、事前の対処が必要です。存在しないノードを定義すると、新規作成されます。しかし、ノードのモニタ設定など細かい設定はできないのでノードは事前に作成しておくことを推奨します。作成済みのノードを使う場合、address パラメータは利用しません。しかし、必須のパラメータのため、省略はできません。設定する値を分かりやすくするため、ノードの IP アドレスを登録することを推奨します。

f5networks.f5_modules.bigip_pool_member モジュールはプールメンバの状態を定義するため、state パラメータは作成（present）、削除（absent）以外も存在します。利用できる state パラメータは Table 7-11 のとおりです。

Table 7-11　f5networks.f5_modules.bigip_pool_member モジュールで利用できる state パラメータ

パラメータ値	説明
present	プールメンバを作成する
absent	プールメンバを削除する
enabled	プールメンバを有効化する
disabled	プールメンバを無効化する
forced_offline	プールメンバをオフラインにする

プールメンバをオフラインにするには「`forced_offline`」を指定します。f5networks.f5_modules.bigip_node モジュールではノードをオフラインにするには「`offline`」のパラメータを指定しました。同じオフラインにする state ですが、指定する値が異なるので注意が必要です。

関　連

▷ プールを管理する（f5networks.f5_modules.bigip_pool モジュール）

https://docs.ansible.com/ansible/latest/collections/f5networks/f5_modules/bigip_pool_module.html

▷ 7-3-3 ノードを作成する

7-3-5　バーチャルサーバを作成する

キーワード

▷ f5networks.f5_modules.bigip_virtual_server モジュール

方　法

バーチャルサーバを作成するプレイブックを紹介します（List 7-13 ）。

List 7-13　バーチャルサーバを作成するプレイブック: ./lib/bigip_virtual_server.yml

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false

  tasks:
    - name: create standard virtual server
      f5networks.f5_modules.bigip_virtual_server:
        state: present
        name: web_server001
        source: 0.0.0.0/0
        destination: '10.100.0.246'
        port: '8080'
        profiles:
          - http_profile
        enabled_vlans:
          - /Common/external
        snat: automap
        type: standard
        pool: http_pool
```

解　説

バーチャルサーバを作成するには、f5networks.f5_modules.bigip_virtual_server モジュールを利用します。

Table 7-12　f5networks.f5_modules.bigip_virtual_server モジュールの主なパラメータ

パラメータ名	説明
name	バーチャルサーバ名を定義
source	バーチャルサーバが接続を許可する送信元ネットワーク、もしくは送信元 IP アドレスを定義
destination	送信先ネットワーク、もしくは送信先 IP アドレスを定義
port	バーチャルサーバの待ち受けポート番号を定義
profiles	プロファイルを定義
enabled_vlans	VLAN を定義
snat	SourceNAT の設定を指定
type	バーチャルサーバのタイプを指定（デフォルト: standard）
pool	プール名を定義

enabled_vlans パラメータは「**パーティション名/VLAN 名**」で定義する必要があります。パーティションを分割していない場合は「**Common/VLAN 名**」で定義します。

f5networks.f5_modules.bigip_virtual_server モジュールは、バーチャルサーバの状態を定義するため、state パラメータは作成（present）、削除（absent）以外も存在します。利用できる state パラメータは Table 7-13 のとおりです。

Table 7-13　f5networks.f5_modules.bigip_virtual_server モジュールで利用できる state パラメータ

パラメータ値	説明
present	バーチャルサーバを作成する
absent	バーチャルサーバを削除する
enabled	バーチャルサーバを有効化する
disabled	バーチャルサーバを無効化する

応　用

応用例として、内部（LAN 側）から外部（インターネット）へのフォワーディングバーチャルサーバを作成するプレイブックを紹介します（List 7-14 ）。

List 7-14　フォワーディングバーチャルサーバ作成するプレイブック: ./lb/bigip_virtual_server_fowarding.yml

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false

  tasks:
    - name: create forwarding virtual server(from internal to external)
      f5networks.f5_modules.bigip_virtual_server:
        state: present
        name: forwarding_virtual_server
        source: '10.100.0.0/26'
        destination: '0.0.0.0'
        mask: '0'
        port: '*'
        ip_protocol: any
        profiles:
          - fastL4
        enabled_vlans:
          - /Common/external
        type: forwarding-ip
```

フォワーディングバーチャルサーバを作成するには type パラメータを「forwarding-ip」に指定します。

関　連

▷ バーチャルサーバを管理する（f5networks.f5_modules.bigip_virtual_server モジュール）

https://docs.ansible.com/ansible/latest/collections/f5networks/f5_modules/bigip_virtual_server_module.html

7-3-6　iRule をバーチャルサーバに適用する

キーワード

▷ f5networks.f5_modules.bigip_virtual_server モジュール

方　法

iRule をバーチャルサーバに適用するには、f5networks.f5_modules.bigip_virtual_server モジュールを利用します（List 7-15 ）。

List 7-15　iRule をバーチャルサーバに適用するプレイブック: ./lb/bigip_irule_attached_virtual_server.yml

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false

  tasks:
    - name: attached irule to virtual server
      f5networks.f5_modules.bigip_virtual_server:
        state: present
        name: web_server001
        irules:
          - _sys_https_redirect

```

解　説

iRule をバーチャルサーバに適用するには、f5networks.f5_modules.bigip_virtual_server モジュールを利用します。iRule を設定する f5networks.f5_modules.bigip_virtual_server モジュールの主なパラメータは Table 7-14 のとおりです。

Table 7-14　iRule を設定する f5networks.f5_modules.bigip_virtual_server モジュールの主なパラメータ

パラメータ名	説明
irules	iRule 名を定義

irules パラメータはリストで登録できます。

関　連

▷ 7-3-5 バーチャルサーバを作成する

7-3-7　プールメンバを無効化にする

キーワード

▷ f5networks.f5_modules.bigip_pool_member モジュール

方　法

プールメンバを無効化にするプレイブックを紹介します（List 7-16 ）。

List 7-16　プールメンバを無効化にするプレイブック: ./lb/bigip_pool_member_offline.yml

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false

  tasks:
    - name: offline pool member
      f5networks.f5_modules.bigip_pool_member:
        state: forced_offline
        pool: http_pool_1
        aggregate:
          - name: web_node001
            address: 10.100.0.135
            port: 80
```

解　説

プールメンバを無効化にするには、f5networks.f5_modules.bigip_pool_member モジュールを利用します。主なパラメータについては、「7-3-4 プールを作成する」を参考にしてください。サーバのメンテナンスなどでプールメンバを一時的に除外するには state パラメータに「disable」もしくは「forced_offline」を指定します。「disable」の場合は、接続しているセッションも含めてすべて無効化します。「forced_offline」の場合は、新規のリクエストは受け付けず、接続しているセッションのみ処理を継続します。作業前提や作業状況に応じて利用する state を選択します。

関　連

▷ 7-3-4 プールを作成する

7-3-8　SSL プロファイルを作成してバーチャルサーバにマッピングする

キーワード

▷ f5networks.f5_modules.bigip_ssl_key_cert モジュール

▷ f5networks.f5_modules.bigip_profile_client_ssl モジュール

▷ f5networks.f5_modules.bigip_virtual_server モジュール

方　法

SSL プロファイルを作成してバーチャルサーバにマッピングするプレイブックを紹介します（List

7-17 ）。

本プレイブックでは、作成済みのバーチャルサーバに対して、クライアント SSL 証明書を設定します。SSL 証明書は事前に発行し、本プレイブックでは Ansible のコントロールノードがアクセスできるディレクトリに格納済みであることを前提とします。

Operation 7-4　SSL 証明書格納フォルダ: ./lb/files

```
files
├── server.crt
└── server.key
```

List 7-17　SSL 証明書を更新するプレイブック: ./lb/bigip_update_ssl_client.yml

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false

  tasks:
    # (1) 秘密鍵と証明書のインポート
    - name: import key and certificate
      f5networks.f5_modules.bigip_ssl_key_cert:
        key_name: sample_key
        key_content: "{{ lookup('file', 'files/server.key') }}"
        cert_name: sample_cert
        cert_content: "{{ lookup('file', 'files/server.crt') }}"

    # (2) SSL プロファイルの作成
    - name: create client ssl profile
      f5networks.f5_modules.bigip_profile_client_ssl:
        name: sslclient01
        cert_key_chain:
          - cert: sample_cert
            key: sample_key

    # (3) SSL プロファイルをバーチャルサーバにマッピング
    - name: update the ssl profile of the virtual server
      f5networks.f5_modules.bigip_virtual_server:
        state: present
        name: web_server001
        port: '443'
        profiles:
```

```
30:           - name: sslclient01
31:             context: client-side
```

解　説

(1) 秘密鍵と証明書のインポート

はじめに、秘密鍵と証明書のインポートを行います。インポートには f5networks.f5_modules.bigip_ssl_key_cert モジュールを利用します。

f5networks.f5_modules.bigip_ssl_key_cert の主なパラメータは Table 7-15 のとおりです。

Table 7-15　f5networks.f5_modules.bigip_ssl_key_cert モジュールの主なパラメータ

パラメータ名	説明
key_name	秘密鍵の名前を定義
key_content	秘密鍵の内容を定義
cert_name	SSL 証明書名を定義
cert_content	SSL 証明書の内容を定義

*_content パラメータには、直接内容を定義します。lookup プラグインを利用しファイルの内容を読み取ることができます。

(2) SSL プロファイルの作成

クライアント SSL プロファイルの作成には、f5networks.f5_modules.bigip_profile_client_ssl モジュールを利用します。

Table 7-16　f5networks.f5_modules.bigip_profile_client_ssl モジュールの主なパラメータ

パラメータ名		説明
name		SSL プロファイルの名前を定義
cert_key_chain		証明書と秘密鍵の内容を定義
	cert	インポートした証明書名を指定
	key	インポートした秘密鍵の名前を指定

中間証明書が必要な場合は、cert_key_chain パラメータの配下に chain パラメータを追加し、インポート済みの中間証明書名を定義します。

(3) SSL プロファイルをバーチャルサーバにマッピング

SSL プロファイルをバーチャルサーバにマッピングするには、f5networks.f5_modules.bigip_virtual_server モジュールを利用します。基本的なパラメータに関しては「7-3-5 バーチャルサーバを作成する」を参考にしてください。

Table 7-17　プロファイルのマッピングで利用する主なパラメータ

パラメータ名		説明
profiles		適用するプロファイルを定義
	name	プロファイル名を指定
	context	プロファイルをサーバ側、クライアント側もしくはすべてにマッピングするかを指定する

作成済みのバーチャルサーバのプロファイルを更新するには、profiles パラメータを利用します。profiles パラメータはリストで定義可能なため、複数のプロファイルをマッピングすることもできます。

関　連

▷ 7-3-2 プロファイルを作成する

▷ 7-3-5 バーチャルサーバを作成する

▷ SSL 証明書をインポートする（f5networks.f5_modules.bigip_ssl_key_cert）

`https://docs.ansible.com/ansible/latest/collections/f5networks/f5_modules/bigip_ssl_key_cert_module.html`

▷ SSL プロファイルを管理する（f5networks.f5_modules.bigip_profile_client_ssl）

`https://docs.ansible.com/ansible/latest/collections/f5networks/f5_modules/bigip_profile_client_ssl_module.html`

7-4　運用管理設定

セッション数の確認やバックアップファイルの取得のように、運用管理で利用するプレイブックについて紹介します。

7-4-1　セッション数を確認する

キーワード

▷ f5networks.f5_modules.bigip_device_info モジュール

方　法

セッション数を確認するプレイブックを紹介します（List 7-18 ）。

List 7-18　セッション数を確認するプレイブック: ./lb/bigip_device_info_ltm_pools.yml

```
---
- hosts: bigip
  connection: ansible.builtin.local
  gather_facts: false
  vars:
    pool: http_pool
  tasks:
    - name: fetch ltm-pool info
      f5networks.f5_modules.bigip_device_info:
        gather_subset:
          - ltm-pools
      register: res_ltm_pool

    - name: debug current sessions
      debug:
        msg: "{{ filter_res.current_sessions }}"
      vars:
        ltm_pools: "{{ res_ltm_pool.ansible_facts.ansible_net_ltm_pools }}"
        filter_res: "{{ ltm_pools | selectattr('name','==',pool) | first }}"
```

解　説

セッション数を確認するには、f5networks.f5_modules.bigip_device_info モジュールを利用します。モジュールの利用方法については「7-1-5 接続確認をする」を参考にしてください。今回は gather_subset パラメータにプールの情報を取得できる「ltm-pools」を指定します。

戻り値を register ディレクティブで指定した変数「res_ltm_pool」に格納します。selectattr フィルタと first フィルタを利用し特定の辞書のみを取り出し、現在のセッション（current_sessions）を確認しています。戻り値は、f5networks.f5_modules.bigip_device_info モジュールの gather_subset パラメータの指定方法によって内容が異なります。詳細は公式ドキュメントを参照してください。

f5networks.f5_modules.bigip_device_info モジュールで必要な情報が取得できない場合、BIG-IP に対して tmsh コマンドを実行できる f5networks.f5_modules.bigip_command モジュールを利用することもできます。f5networks.f5_modules.bigip_command モジュールはコマンドをシンプルに送信するモジュールです。

関　連

▷ 2-3-1 辞書のリストから特定条件の要素を抽出する
▷ 7-1-5 接続確認をする
▷ デバイス情報を収集する（f5networks.f5_modules.bigip_device_info モジュール）
https://docs.ansible.com/ansible/latest/collections/f5networks/f5_modules/bigip_device_info_module.html

7-4-2　バックアップを作成する

キーワード

▷ f5networks.f5_modules.bigip_ucs_fetch モジュール

方　法

バックアップファイルを作成して、コントロールノードにダウンロードするプレイブックを紹介します（List 7-19）。

```
1: ---
2: - hosts: bigip
3:   connection: ansible.builtin.local
4:   gather_facts: false
5:   vars:
6:     backup_ucs: "{{ now(False, '%Y%m%d_%H%M%S') }}.ucs"
7:
```

```
 8:  tasks:
 9:    - name: create and download ucs
10:      f5networks.f5_modules.bigip_ucs_fetch:
11:        src: "{{ backup_ucs }}"
12:        create_on_missing: true
13:        dest: "{{ playbook_dir }}/files/{{ backup_ucs }}"
```

解　説

バックアップファイルを作成するには、f5networks.f5_modules.bigip_ucs_fetch モジュールを利用します。

Table 7-18　f5networks.f5_modules.bigip_ucs_fetch モジュールの主なパラメータ

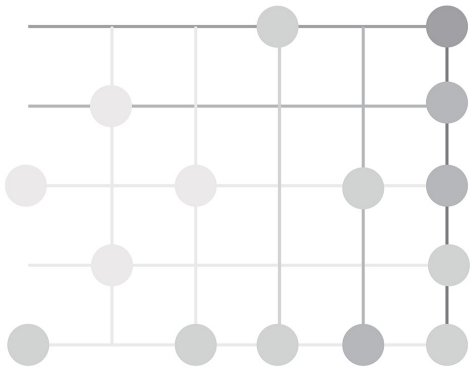
パラメータ名	説明
src	ダウンロードをするバックアップファイルのパスを定義
create_on_missing	src パラメータで指定しているファイルが存在しない場合、新規作成する
fail_on_missing	src パラメータで指定しているファイルが存在しない場合、失敗にする
dest	バックアップファイルを格納するコントロールノード側のパスを定義

f5networks.f5_modules.bigip_ucs_fetch モジュールは、バックアップファイルの作成とコントロールノードへのダウンロードが可能です。create_on_missing パラメータはデフォルトで「true」のため、省略をした場合でもバックアップファイルの新規作成が可能です。dest パラメータは必須です。バックアップファイルの作成のみはできないモジュールなので、使用には注意してください。

関　連

▷ バックアップを管理する（f5networks.f5_modules.bigip_ucs_fetch モジュール）
https://docs.ansible.com/ansible/latest/collections/f5networks/f5_modules/bigip_ucs_fetch_module.html

第８章
SDN



Ansible はさまざまな SDN（Software Defined Network）製品にも対応しています。SDN コンポーネントは設定が抽象化されて簡単になったメリットもあります。一方で GUI による設定となる場合が多く、運用者の負担が増えることも多いのが実情です。

運用効率化のためにスクリプトを作成する手段もありますが、プログラミング言語の学習はハードルが高いと思われる方も多いでしょう。Ansible を活用することで比較的簡単に自動化処理を実装でき、高い効果が期待できます。

本章では SDN 製品の Cisco ACI（Application Centric Infrastructure）を対象に運用時の課題を解決するプレイブックを紹介します。

8-1　SDN の対応概要

各プレイブックの紹介に先立ち、Ansible が SDN に対してできることや環境の準備方法、各プレイブックの前提となる構成を説明します。

SDN の代表的な製品である以下の 3 つはコレクションが公開されているため、Ansible による自動化を始めやすいのもポイントです。

- VMware NSX-T
- Cisco ACI
- Cisco DNA Center

SDN と Ansible を組み合わせる場合、コントローラへの設定のみでファブリックにも設定されます。そのため、Ansible のインベントリとして登録する機器は、コントローラのみでよいといったメリットがあります。

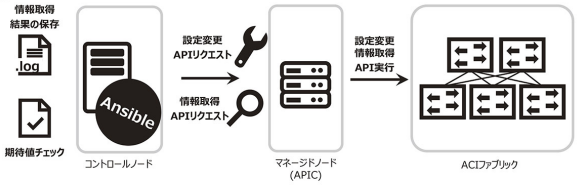
8-1-1　対応範囲

Ansible が Cisco ACI に対して、自動化できる内容を説明します。

■ 自動化できること

Cisco ACI のコントローラである APIC（Application Policy Infrastructure Controller）に対して Ansible から操作することで、ACI ファブリックと呼ばれる、スパインスイッチおよびリーフスイッチで構成された機器群に対して設定の参照や変更が可能です（Figure 8-1）。

Figure 8-1　自動化でできること



Cisco ACI に対応する cisco.aci コレクションはバージョン 2.0.0 時点でモジュールは 86 個あります。これらのモジュールを組み合わせるだけでもアンダーレイ設定やオーバーレイ設定など多くの作業が自動化可能です。

さらに特徴的なモジュールとして cisco.aci.aci_rest があります。cisco.aci.aci_rest は APIC に対して直接 REST API のリクエストを送信できるモジュールです。そのため、現時点でモジュールで実装されていない APIC の作業に関しても Ansible から実現できます。

8-1-2 環境の準備

■ コントロールノード側の準備

コントロールノード側では 4 つの作業を実施します。

(1) cisco.aci コレクションのインストール

本章では pip で ansible 4.2.0 をインストール時に同梱されている、cisco.aci コレクションのバージョン 2.0.0 を利用します。

同梱バージョン以外を使いたい場合は、cisco.aci コレクションを「`ansible-galaxy collection install`」コマンドでインストールします（Operation 8-1）。

Operation 8-1 cisco.aci コレクションのインストールコマンド実行例

```
$ ansible-galaxy collection install cisco.aci
```

(2) Python ライブラリのインストール

cisco.aci コレクションのモジュールを使う際に、基本的には Python のライブラリは必要ありません。ただし、ユーザ作成で利用する cisco.aci.aci_user モジュールを利用するときのみ `python-dateutil` が必要です（Operation 8-2）。

Operation 8-2 Python ライブラリのインストールコマンドの実行例

```
$ pip install python-dateutil
```

(3) 環境変数の設定

続いて、APIC にログインするための認証情報を環境変数として設定します（Operation 8-3）。環境

変数を利用することでモジュール利用時にユーザ名やパスワードなどの指定が不要になり、可読性が向上します。

Operation 8-3　環境変数の設定実行例

```
# 実行コマンド（変数は環境に応じて設定）
$ export ACI_HOST='IPaddress'
$ export ACI_USERNAME='username'
$ export ACI_PASSWORD='password'
$ export ACI_PORT=443
$ export ACI_VALIDATE_CERTS='False'
```

環境変数を設定した場合のプレイブック（List 8-1）と設定をしないプレイブック（List 8-2）を比較します。記述量が減り、処理内容が分かりやすくなります。

List 8-1　環境変数利用時のプレイブック

```
1: ---
2: - hosts: aci
3:   gather_facts: false
4:   connection: ansible.builtin.local
5:
6:   tasks:
7:     - name: fetch aci system
8:       cisco.aci.aci_system:
```

List 8-2　環境変数を利用しないプレイブック

```
 1: ---
 2: - hosts: aci
 3:   gather_facts: false
 4:   connection: ansible.builtin.local
 5:
 6:   tasks:
 7:     - name: fetch aci system
 8:       cisco.aci.aci_system:
 9:         host: XX.XX.XX.XX
10:         username: user
11:         password: password
12:         validate_certs: false
```

(4) Python インタープリタの設定

インストールしている Ansible の環境によっては、モジュール実行時に利用する Python インタープリタの設定も必要です。

設定には ansible_python_interpreter 変数か、ansible.cfg の defaults セクションの「interpreter_python」を利用します。たとえば「ansible」を「/home/ansible/myvenv」という venv にインストールした場合は「/home/ansible/myvenv/bin/python」を指定します（List 8-3）。

ほかには、venv のパスを直接指定するのではなく、マジック変数「ansible_playbook_python」を指定する方法もあります。

List 8-3　ansible.cfg で Python インタープリタを設定する例

```
1: [defaults]
2: interpreter_python =/home/ansible/myvenv/bin/python
```

■ マネージドノード（APIC）側の準備

マネージドノードである APIC（Application Policy Infrastructure Controller）には、Ansible を利用する設定は必要ありません。必要に応じて、Ansible でアクセスするためのユーザを作成してください。

APIC への接続時には、2 種類の認証方式があります（Table 8-1）。

Table 8-1　接続時の認証方式

認証方式	説明
証明書認証	X.509 証明書と秘密鍵を用いた認証
パスワード認証	ユーザとパスワードを用いた認証

パスワード認証は非常に簡単で手軽に始められるのがメリットです。ただし APIC に対して処理が高速かつ連続で行われると、DoS 攻撃対策のセッションの上限に達し、処理が失敗することがあります。そのため、ログインとログアウトが不要になり、セッション管理がなくなる証明書認証を採用する方が望ましいと言えます。2 つの認証方式についての詳細は公式ドキュメントの「ACI authentication[*1]」を参照してください。パスワードや秘密鍵の管理には ansible-valut を利用し、暗号化して管理することを推奨します。本書では手軽にできるパスワード認証方式を採用します。

＊1　ACI authentication
http://docs.ansible.com/ansible/latest/scenario_guides/guide_aci.html#aci-authentication

8-1-3　cisco.aci コレクションのモジュールの基本仕様

cisco.aci コレクションのモジュールの基本仕様について紹介します。プレイブックで指定されたパラメータから REST API の URL とペイロードを生成し、state パラメータの指定に対応する HTTP メソッドでリクエストを送信します（Table 8-2）。

Table 8-2　メソッドと state パラメータの関係

HTTP メソッド	モジュールで指定する state パラメータの値	説明
GET	query	情報取得
POST	present	設定追加
DELETE	absent	設定削除

他の Ansible モジュールの state パラメータは設定（present）と削除（absent）で構成されていることが多いのですが、cisco.aci コレクションの aci モジュールでは情報取得の query も用意されています。

■ モジュール一覧と分類

cisco.aci コレクションのモジュール[*2]は「aci_*」という命名ルールで作成されています。また、モジュール名から処理内容がある程度予想ができます。たとえば、Cloud ACI を対象としたモジュールは、cisco.aci.aci_cloud_* というモジュール名が付けられています。

特定のリソースを紐付ける場合には、cisco.aci.aci_*_to_* のモジュールを探しましょう。

■ 冪等性

cisco.aci コレクションのモジュールは、基本的には冪等性が担保されます。処理の初めに情報を取得し、すでに設定が存在していれば追加や変更を行いません。ただし例外的に、cisco.aci.aci_aaa_user モジュールで aaa_password パラメータを利用する場合は冪等性が担保されません。

■ 接続時に利用する特別な変数

APIC への接続時には、コネクションプラグインを ansible.builtin.local、もしくはタスクの delegate_to ディレクティブに localhost を指定して実行する必要があります。プレイで定義するタス

＊2　ACI コレクションのモジュール一覧
http://docs.ansible.com/ansible/latest/collections/cisco/aci/index.html

クが APIC への作業のみの場合は、コネクションプラグインを ansible.builtin.local に指定します（Table 8-3）。

Table 8-3　接続に利用する特別な変数

変数名	説明
ansible_connection	利用するコネクションプラグイン。 Cisco ACI では「ansible.builtin.local」を利用する

接続先や認証情報の指定は「8-1-2 環境の準備」で説明したとおり、環境変数を利用します。

Column　ドキュメントにある「delegate_to: localhost」の補足

公式ドキュメントの cisco.aci コレクションのサンプルプレイブックでは、ほぼすべてのサンプルのタスクに「delegate_to: localhost」のディレクティブを設定しています。これは実行先をローカルホストにするのと同じ意味になり、「connection: ansible.builtin.local」を設定している場合は不要です。ドキュメントでは、なるべくタスク単位でコピーアンドペーストして使えるようにするため、あえて「delegate_to: localhost」を指定していると考えられます。

プレイに「connection: ansible.builtin.local」 を指定する場合、プレイ内のすべてのタスクの接続方式が local になる点は注意が必要です。

8-1-4　本章の前提構成

8-2 節以降で掲載する各プレイブックの前提となる環境や、インベントリ、変数定義ファイルについて説明します。

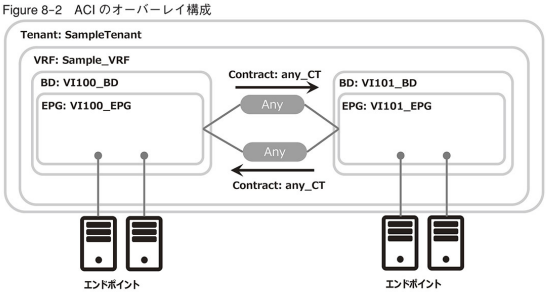
■ 環境

本章で利用する環境について紹介します（Figure 8-2）。

APIC のバージョンは 4.1（1k）を利用します。ネットワークセントリック（ネットワーク中心）な設計でブリッジドメイン、エンドポイントグループ、VLAN がそれぞれ 1 対 1 でマッピングされた比較的シンプルな運用を想定しています。

本章の解説では、ACI の構築が完了し、オペレータによる運用作業を開始している状態を想定しています。

Figure 8-2　ACI のオーバーレイ構成



■ インベントリと変数定義ファイル

インベントリと変数定義ファイルについて紹介します（List 8-4）。cisco.aci コレクションのモジュールはローカルコネクションプラグインを利用するため、接続先はローカルホストです。また、認証情報は環境変数で設定しているため、インベントリはグループ変数定義ファイルに定義した変数を関連付けるために作成します。

List 8-4　インベントリファイル: ./sdn/inventory.ini

```
1: [aci]
2: apic
```

構築済みのテナント、アプリケーションプロファイル（ap）、VRF、アタッチャブルエンティティプロファイル（aep）、ドメインをグループ変数定義ファイルに定義します（List 8-5）。

List 8-5　グループ変数定義ファイル: ./sdn/group_vars/aci.yml

```
1: ---
2: tenant: SampleTenant
3: ap: SampleApp_AP
4: vrf: Sample_VRF
5: aep: Sample_AAEP
6: domain: Sample_phydom
```

8-1-5 接続確認をする

環境変数などに設定した認証情報を確認するため、APIC への接続確認を行います（List 8-6）。

APIC への接続確認には、cisco.aci.aci_system モジュール[*3]を使うことを推奨します。cisco.aci.aci_system モジュールは ACI のシステム情報を参照するモジュールです。

List 8-6 接続確認用プレイブック: ./sdn/aci_system.yml

```
1: ---
2: - hosts: aci
3:   gather_facts: false
4:   connection: ansible.builtin.local
5:
6:   tasks:
7:     - name: fetch aci system
8:       cisco.aci.aci_system:
```

cisco.aci.aci_system モジュールは利用できる HTTP メソッドは GET のみ（state パラメータは query のみ指定可能）です。そのため、初めに実施するモジュールとして、疎通性および認証を確認するのに最適です。今回のサンプルでは接続確認のために利用しています。戻り値を register ディレクティブで指定するレジスタ変数に格納し、スパインやリーフ、APIC のシステム情報の確認も可能です。

8-2 アンダーレイ設定

Cisco ACI におけるアンダーレイ設定を行うためのプレイブックを紹介します。アンダーレイ作業を代表して、機器やサーバ接続時に必要となるインターフェイスポリシーとインターフェイスプロファイルの作成などを解説します。

8-2-1 インターフェイスポリシーを作成しインターフェイスポリシーグループと関連付ける

キーワード

▷ cisco.aci.aci_interface_policy_link_level モジュール

▷ cisco.aci.aci_interface_policy_cdp モジュール

▷ cisco.aci.aci_interface_policy_leaf_policy_group モジュール

＊3 ACI のシステム情報を確認する（cisco.aci.aci_system モジュール）
http://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_system_module.html

方　法

インターフェイスポリシーとインターフェイスポリシーグループの作成方法を紹介します。新たな接続方式（スピードや CDP 設定など）で機器と接続するために必要なオペレーションです。モジュールを組み合わせることで簡単に作成できます。

List 8-7 の例では、リンクポリシー 1G 用と CDP 有効化ポリシーの作成後に、追加したポリシーをマッピングしたポリシーグループを作成します。

List 8-7　インターフェイスポリシーとインターフェイスポリシーグループを作成するプレイブック: ./sdn/aci_interface_policy.yml

```
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local
  vars:
    policy_group: Sample_APG

  tasks:
    # (1) リンクレベルポリシーの作成（1G 用）
    - name: create interface policy link level
      cisco.aci.aci_interface_policy_link_level:
        state: present
        link_level_policy: 1GigAuto
        auto_negotiation: true
        speed: 1G

    # (2) CDP ポリシーの作成（有効化用）
    - name: create interface policy cdp
      cisco.aci.aci_interface_policy_cdp:
        state: present
        cdp_policy: CDP_Enable
        admin_state: true

    # (3) ポリシーグループの作成（CDP とリンクレベルポリシーを含めたグループ）
    - name: create interface policy group leaf access
      cisco.aci.aci_interface_policy_leaf_policy_group:
        state: present
        lag_type: leaf
        policy_group: "{{ policy_group }}"
        link_level_policy: 1GigAuto
        cdp_policy: CDP_Enable
        aep: "{{ aep }}"
```

解　説

(1) リンクレベルポリシーの作成（1G 用）

リンクレベルポリシーの作成には、cisco.aci.aci_interface_policy_link_level モジュールを利用します。

cisco.aci.aci_interface_policy_link_level モジュールの主なパラメータは Table 8-4 のとおりです。

Table 8-4　cisco.aci.aci_interface_policy_link_level モジュールの主なパラメータ

パラメータ名	説明
link_level_policy	リンクレベルポリシー名を定義
auto_negotiation	オートネゴシエーションの有効化、無効化を指定（デフォルト: false ）
speed	スピードを指定（デフォルト: inherit ）

speed パラメータはデフォルトで inherit が指定されているため変更します。指定できる値は 100M、1G、10G、40G などです。今回は 1G 用のポリシーが必要なため 1G を指定しています。

(2) CDP ポリシーの作成（有効化用）

CDP ポリシーの作成には、cisco.aci.aci_interface_policy_cdp モジュールを利用します。

cisco.aci.aci_interface_policy_cdp モジュールの主なパラメータは Table 8-5 のとおりです。

Table 8-5　cisco.aci.aci_interface_policy_cdp モジュールの主なパラメータ

パラメータ名	説明
cdp_policy	CDP ポリシー名を定義
admin_state	CDP の有効化、無効化を指定（デフォルト: true）

有効化、無効化を定義する admin_state のパラメータはデフォルトが true（有効化）のため、パラメータを省略しても実行結果は変わりません。しかし、パラメータを明示的に指定することで、後からプレイブックを見直した際にも設定が分かりやすくなります。そのため、デフォルトの値から変更しない場合でも意図的にパラメータを記載することを推奨します。

(3) ポリシーグループの作成（CDP とリンクレベルポリシーを含めたグループ）

ポリシーグループを作成する cisco.aci.aci_interface_policy_leaf_policy_group モジュールは、利用するポリシーによってパラメータの組み合わせが異なります。作成した CDP ポリシーとリンクレベルポリシーを組み合わせたポリシーグループを作成します（Table 8-6）。

Table 8-6　cisco.aci.aci_interface_policy_leaf_policy_group モジュールの主なパラメータ

パラメータ名	説明
lag_type	作成するリーフポリシーグループのタイプを定義
policy_group	ポリシーグループ名を定義
cdp_policy	CDP ポリシーを定義
link_level_policy	リンクレベルポリシーを定義
aep	対象 AEP を定義

lag_type パラメータは、利用するリンクアグリゲーションの種類により指定するキーワードが異なります。今回のアクセスポートポリシーに合わせて leaf を指定します。

Column　Policy 関連モジュール

ポリシーに関連するモジュールは cisco.aci.aci_interface_policy_* の命名ルールになっています。今回利用した、リンクレベルと CDP 以外にも多くのモジュールが用意されています。運用に合わせて必要なモジュールを組み合わせて利用できます。

- cisco.aci.aci_interface_policy_fc
- cisco.aci.aci_interface_policy_leaf_breakout_port_group
- cisco.aci.aci_interface_policy_leaf_profile
- cisco.aci.aci_interface_policy_lldp
- cisco.aci.aci_interface_policy_ospf
- cisco.aci.aci_interface_policy_port_security
- cisco.aci.aci_interface_policy_l2
- cisco.aci.aci_interface_policy_leaf_policy_group
- cisco.aci.aci_interface_policy_link_level
- cisco.aci.aci_interface_policy_mcp
- cisco.aci.aci_interface_policy_port_channel

応　用

ポリシーの組み合わせを変えて VPC 用のポリシーグループを作成します。Portchannel のポリシーを作成後、「`lag_type: node`」でインターフェイスプロファイルを作成します（List 8-8）。

List 8-8　VPC 用インターフェイスポリシー作成プレイブック: ./sdn/aci_interface_policy_vpc.yml

```
1: ---
2: - hosts: aci
3:   gather_facts: false
4:   connection: ansible.builtin.local
5:   vars:
```

```
    policy_group: Sample_VPC
    lacp_policy: LACP_Active

tasks:
    # (1) ポートチャネル用ポリシーの作成
    - name: create interface policy port channel
      cisco.aci.aci_interface_policy_port_channel:
        state: present
        port_channel: "{{ lacp_policy }}"
        mode: active

    # (2) インターフェイスポリシーグループの作成
    - name: create interface policy group vpc
      cisco.aci.aci_interface_policy_leaf_policy_group:
        state: present
        lag_type: node
        policy_group: "{{ policy_group }}"
        port_channel_policy: "{{ lacp_policy }}"
        link_level_policy: 1GigAuto
        cdp_policy: CDP_Enable
        aep: "{{ aep }}"
```

cisco.aci.aci_interface_policy_leaf_policy_group モジュールに port_channel_policy パラメータを追加し、作成したポートチャネル用ポリシーを指定します。

関　連

▷ リンクポリシーを管理する（aci_interface_policy_link_level モジュール）

http://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_interface_policy_link_level_module.html

▷ CDP ポリシーを管理する（aci_interface_policy_cdp モジュール）

http://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_interface_policy_cdp_module.html

▷ ポートチャネルポリシーを管理する（aci_interface_policy_port_channel モジュール）

http://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_interface_policy_port_channel_module.html

▷ ポリシーグループを管理する（aci_interface_policy_leaf_policy_group モジュール）

http://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_interface_policy_leaf_policy_group_module.html

8-2-2　インターフェイスプロファイルを作成する

キーワード

▷ cisco.aci.aci_interface_policy_leaf_profile モジュール

▷ cisco.aci.aci_access_port_to_interface_policy_leaf_profile モジュール

▷ cisco.aci.aci_access_port_block_to_access_port モジュール

方　法

インターフェイスポリシーグループとアクセスポートをマッピングしたインターフェイスプロファイルの作成と、リーフプロファイルへのマッピング方法を紹介します。リーフプロファイルは作成済みのものを利用します（List 8-9）。

List 8-9　インターフェイスプロファイルを作成しリーフプロファイルと関連付けるプレイブック: ./sdn/aci_interface_profile.yml

```
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local
  vars:
    interface_profile: Lf101_IntProf
    interface_selector: Lf101_IntSele
    policy_group: Sample_APG
    leaf_profile: Leaf101_SwitchProfile
    access_ports:
      - 30
      - 33
      - 35

  tasks:
    # (1) インターフェイスプロファイルの作成
    - name: create leaf interface profile
      cisco.aci.aci_interface_policy_leaf_profile:
        state: present
        type: leaf
        interface_profile: "{{ interface_profile }}"

    # (2) インターフェイスセレクタの作成
    - name: create interface selector
      cisco.aci.aci_access_port_to_interface_policy_leaf_profile:
        state: present
```

```
27:         interface_profile: "{{ interface_profile }}"
28:         access_port_selector: "ethernet1_{{ item }}"
29:         policy_group: "{{ policy_group }}"
30:       loop: "{{ access_ports }}"
31:
32:     # (3) インターフェイスセレクタにアクセスポートをマッピング
33:     - name: bind access port to interface selector
34:       cisco.aci.aci_access_port_block_to_access_port:
35:         interface_profile: "{{ interface_profile }}"
36:         access_port_selector: "ethernet1_{{ item }}"
37:         port_blk: "blk_{{ item }}"
38:         from_port: "{{ item }}"
39:         to_port: "{{ item }}"
40:         state: present
41:       loop: "{{ access_ports }}"
42:
43:     # (4) リーフプロファイルへのマッピング
44:     - name: bind interface profile to switch policy
45:       cisco.aci.aci_interface_selector_to_switch_policy_leaf_profile:
46:         state: present
47:         leaf_profile: "{{ leaf_profile }}"
48:         interface_selector: "{{ interface_profile }}"
```

解　説

(1) インターフェイスプロファイルの作成

インターフェイスのプロファイルの作成には、cisco.aci.aci_interface_policy_leaf_profile モジュールを利用します（Table 8-7）。

Table 8-7　cisco.aci.aci_interface_policy_leaf_profile モジュールの主なパラメータ

パラメータ名	説明
type	プロファイルのタイプを指定
interface_profile	インターフェイスプロファイル名を定義

type パラメータで指定可能な値は leaf と fex です、本タスクではサーバ接続を想定して leaf で作成します。

(2) インターフェイスセレクタの作成

インターフェイスのプロファイルにアクセスポートをマッピングするには、インターフェイスセレクタが必要です。インターフェイスセレクタの作成は cisco.aci.aci_access_port_to_interface_policy_leaf_profile モジュールを利用します（Table 8-8）。

Table 8-8　cisco.aci.aci_access_port_to_interface_policy_leaf_profile モジュールの主なパラメータ

パラメータ名	説明
interface_profile	インターフェイスプロファイル名を定義
access_port_selector	アクセスポートセレクタを定義
policy_group	ポリシーグループを定義

主なパラメータには掲載していませんが、アクセスポートセレクタの作成後にアクセスポートをマッピングする from_card、from_port などのパラメータがあります。こちらは削除される予定のため、利用せずに cisco.aci.aci_access_port_block_to_access_port モジュールで設定することを推奨します。

(3) インターフェイスセレクタにアクセスポートをマッピング

作成したアクセスポートセレクタにアクセスポートをマッピングするには、cisco.aci.aci_access_port_block_to_access_port モジュールを利用します（Table 8-9）。

Table 8-9　cisco.aci.aci_access_port_block_to_access_port モジュールの主なパラメータ

パラメータ名	説明
access_port_selector	アクセスポートセレクタを定義
interface_profile	インターフェイスポリシーを定義
policy_group	ポリシーグループを定義
port_blk	アクセスポートブロックの名前を定義
from_port	マッピング範囲の開始ポートを定義
to_port	マッピング範囲の終了ポートを定義

本サンプルでは from_port と to_port パラメータには同じ値が入り、1 ポートずつマッピングしています。「`from_port: 1`」、「`to_port: 10`」のように定義することで範囲登録できます。

(4) リーフプロファイルへのマッピング

Table 8-10　cisco.aci.aci_interface_selector_to_switch_policy_leaf_profile モジュールの主なパラメータ

パラメータ名	説明
leaf_profile	リーフプロファイルを定義
interface_selector	インターフェイスプロファイルを定義

cisco.aci.aci_interface_selector_to_switch_policy_leaf_profile モジュールのパラメータ名は名前と実体がずれているため注意が必要です。leaf_profile パラメータはリーフプロファイル（スイッチプロファイルと呼ぶ場合もある）を定義しますが、interface_selector パラメータはインターフェイスプロファイルを定義します。分かりにくい場合は、パラメータ名のエイリアス機能（別名）を利用し interface_selector を interface_profile_name に変更して使うのがよいでしょう（List 8-10）。

List 8-10　エイリアスを利用したタスク

```
    # エイリアス利用時の表記方法
    - name: bind interface profile to switch policy
      cisco.aci.aci_interface_selector_to_switch_policy_leaf_profile:
        state: present
        leaf_profile: "{{ leaf_profile }}"
        interface_profile_name: "{{ interface_profile }}"
```

応　用

応用として VPC 用のインターフェイスプロファイルの作成とリーフプロファイルへのマッピングを紹介します。リーフプロファイルは作成済みのものを利用します（List 8-11）。

List 8-11　VPC 用プロファイルの作成とリーフプロファイルへのマッピング:
./sdn/aci_interface_profile_vpc.yml

```
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local
  vars:
    interface_profile: Lf101-102_IntProf
    interface_selector: Lf101-102_IntSele
    policy_group: Sample_VPC
    leaf_profile: Leaf101-102_SwitchProfile
    access_ports:
      - 25
```

```
      - 26

  tasks:
    - name: create leaf interface profile
      cisco.aci.aci_interface_policy_leaf_profile:
        state: present
        type: leaf
        interface_profile: "{{ interface_profile }}"

    - name: create interface selector
      cisco.aci.aci_access_port_to_interface_policy_leaf_profile:
        state: present
        interface_profile: "{{ interface_profile }}"
        access_port_selector: "ethernet1_{{ item }}"
        policy_group: "{{ policy_group }}"
        interface_type: vpc  # VPC 用のプロファイルの作成
      loop: "{{ access_ports }}"

    - name: bind access port to interface selector
      cisco.aci.aci_access_port_block_to_access_port:
        interface_profile: "{{ interface_profile }}"
        access_port_selector: "ethernet1_{{ item }}"
        port_blk: "blk_{{ item }}"
        from_port: "{{ item }}"
        to_port: "{{ item }}"
        state: present
      loop: "{{ access_ports }}"

    - name: bind interface profile to switch policy
      cisco.aci.aci_interface_selector_to_switch_policy_leaf_profile:
        state: present
        leaf_profile: "{{ leaf_profile }}"
        interface_selector: "{{ interface_profile }}"
```

VPC 用プロファイル作成時のポイントは cisco.aci.aci_access_port_to_interface_policy_leaf_profile の interface_type のパラメータです。interface_type パラメータを vpc に指定することで VPC 用のプロファイルが作成されます。

関　連

▷ インターフェイスプロファイルを管理する（aci_interface_policy_leaf_profile モジュール）

http://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_interface_policy_leaf_profile_module.html

▷ インターフェイスセレクタを管理する（aci_access_port_to_interface_policy_leaf_profile モジュール）
http://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_access_port_to_interface_policy_leaf_profile_module.html
▷ インターフェイスセレクタにアクセスポートをマッピングする（cisco.aci.aci_access_port_block_to_access_port モジュール）
http://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_access_port_block_to_access_port_module.html
▷ リーフプロファイルとインターフェイスプロファイルをマッピングする（cisco.aci.aci_access_port_to_interface_policy_leaf_profile モジュール）
http://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_interface_selector_to_switch_policy_leaf_profile_module.html

8-3 オーバーレイ設定

Cisco ACI におけるオーバーレイ設定を行うためのプレイブックを紹介します。オーバーレイ設定の中でもブリッジドメイン、エンドポイントグループ、コントラクトなど運用作業で扱うことの多い設定にフォーカスした内容を解説します。

8-3-1 ブリッジドメイン（BD）を作成する

キーワード

▷ cisco.aci.aci_bd モジュール

方 法

ブリッジドメイン（BD）は cisco.aci.aci_bd モジュールで作成します。IP ルーティングが有効化されたブリッジドメインを作成するプレイブックを紹介します（List 8-12）。

List 8-12 ブリッジドメインを作成するプレイブック: ./sdn/aci_bd.yml

```
1: ---
2: - hosts: aci
3:   gather_facts: false
4:   connection: ansible.builtin.local
5:   vars:
```

```
    bd: Vl500_BD

tasks:
  # (1) ブリッジドメイン作成
  - name: create bridge domain
    cisco.aci.aci_bd:
      state: present
      tenant: "{{ tenant }}"
      bd: "{{ bd }}"
      vrf: "{{ vrf }}"
      bd_type: ethernet
      enable_routing: true
      ip_learning: true
      arp_flooding: true
      l2_unknown_unicast: flood
      l3_unknown_multicast: flood
      endpoint_move_detect: garp
      endpoint_clear: true
```

解　説

cisco.aci.aci_bd モジュールの主なパラメータは Table 8-11 のとおりです。

Table 8-11　cisco.aci.aci_bd モジュールの主なパラメータ

パラメータ名	説明
bd	ブリッジドメイン名を定義
vrf	VRF 名を定義
bd_type	ブリッジドメインのタイプを指定（デフォルト: ethernet）
enable_routing	ルーティング機能の有効化、無効化を指定（デフォルト: false）
ip_learning	エンドポイント IP の学習の有効化、無効化を指定（デフォルト: false）
arp_flooding	ARP トラフィックのフラッディングの有効化、無効化を指定（デフォルト: false）
l2_unknown_unicast	宛先不明の L2 通信に使用する転送方法を指定（デフォルト: proxy）
l3_unknown_multicast	宛先不明のマルチキャストに使用する転送方法を指定（デフォルト: flood）
endpoint_move_detect	エンドポイントの移動を検出するための GARP（Gratuitous ARP）の有効化、無効化を指定（デフォルト: garp）
endpoint_clear	エンドポイントがクリアされた際、すべてのリーフでエンドポイントをクリアするか指定（デフォルト: false）

ブリッジドメインの作成前には、テナントと VRF が必須です。ルーティング機能の有効化や宛先不明な L2、L3 通信を受けた際の動作など、指定する項目が多く存在します。設計に合わせてパラメー

タを調整します。

「state: query」を利用することでブリッジドメインの設定状況の確認もできます。

関 連

▷ ブリッジドメインを管理する（cisco.aci.aci_bd モジュール）

https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_bd_module.html

8-3-2 ブリッジドメイン（BD）にサブネットを設定する

キーワード

▷ cisco.aci.aci_bd_subnet モジュール

方 法

ブリッジドメインにサブネットを設定するには、cisco.aci.aci_bd_subnet モジュールを利用します。サブネットを設定することで接続しているエンドポイントのゲートウェイにできます（List 8-13）。

List 8-13 ブリッジドメインにサブネットを設定するプレイブック: ./sdn/aci_bd_subnet.yml

```
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local
  vars:
    bd: Vl500_BD
    gateway: 192.168.0.1/24

  tasks:
    # (1) ブリッジドメインにサブネットを設定
    - name: add a subnet to bd
      cisco.aci.aci_bd_subnet:
        state: present
        tenant: "{{ tenant }}"
        bd: "{{ bd }}"
        gateway: "{{ gateway | ansible.netcommon.ipaddr('address') }}"
        mask: "{{ gateway | ansible.netcommon.ipaddr('prefix') }}"
        scope: private
```

解　説

cisco.aci.aci_bd_subnet モジュールの主なパラメータは Table 8-12 のとおりです。

Table 8-12　cisco.aci.aci_bd_subnet モジュールの主なパラメータ

パラメータ名	説明
bd	ブリッジドメイン名を定義
tenant	テナント名を定義
vrf	VRF 名を定義
gateway	ゲートウェイ IP 定義
mask	サブネットマスクを定義
scope	サブネットのスコープを指定（デフォルト: private）

作成済みのブリッジドメインに対してサブネットを設定するモジュールです。そのため、前提条件としてユニキャストルーティングが有効化されているブリッジドメインが作成されている必要があります（List 8-14）。

mask パラメータはプレフィックス長形式で指定する必要があります。本項のプレイブックでは、インプットの 192.168.0.1/24 から ipaddr フィルタの prefix を利用し、24 だけ抜き出しています。

scope パラメータはサブネットの特性により指定する値が異なります。ルートリークが必要なサブネットには shared を指定します。リスト形式でも指定できるため複数の値を登録できます。ただし、private と public は併用できません。

List 8-14　ルートリークが必要なサブネットのパラメータ定義方法

```
tasks:
  # (1) ブリッジドメインにサブネットを設定
  - name: add a subnet to bd
    cisco.aci.aci_bd_subnet:
      state: present
      tenant: "{{ tenant }}"
      bd: "{{ bd }}"
      gateway: "{{ gateway | ansible.netcommon.ipaddr('address') }}"
      mask: "{{ gateway | ansible.netcommon.ipaddr('prefix') }}"
      scope:
        - private
        - shared # ルートリーク用のスコープ
```

応　用

cisco.aci.aci_bd_subnet モジュールの「`state: query`」を利用すると、テナント内に設定されている

サブネットの情報をすべて取得できます。json_query と flatten フィルタを利用し、サブネットのみを抜き出し表示します（List 8-15）。

List 8-15　テナント内のサブネットを取得するプレイブック: ./sdn/aci_bd_subnet_advanced.yml

```
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local

  tasks:
    - name: get the target subnet configured for the tenant
      cisco.aci.aci_bd_subnet:
        state: query
        tenant: "{{ tenant }}"
      register: result

    - name: debug all subnet
      ansible.builtin.debug:
        msg: "{{ result['current'][0] | community.general.json_query(query_filter) |⇒
ansible.builtin.flatten(levels=1) }}"
      vars:
        query_filter: fvTenant.children[*].fvBD.children[*].fvSubnet.attributes.ip

```

この例では cisco.aci.aci_bd_subnet モジュールで利用しているパラメータは tenant のみです。この場合テナントの配下にいる VRF すべてが query（情報取得）の対象になります。VRF まで絞り込んで query をしたい場合は vrf のパラメータを追加することで絞り込みができます。

Operation 8-4　実行結果

```
PLAY [aci] *********************************************************************************

TASK [get the target subnet configured for the tenant]*********************************
ok: [apic]

TASK [debug all subnet] ********************************************************************
ok: [apic] => {
    "msg": [
        "192.168.1.1/24",
        "192.168.0.1/24"
    ]
```

すべてのサブネットの情報が取得できるため、作成予定のサブネットが設定されていないか、設定

後に想定どおりサブネットが登録されているか、の確認に利用できます。

関　連

▷ BD のサブネットを管理する（cisco.aci.aci_bd_subnet モジュール）
https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_bd_subnet_module.html

8-3-3　エンドポイントグループ（EPG）を作成する

キーワード

▷ cisco.aci.aci_epg モジュール
▷ cisco.aci.aci_epg_to_domain モジュール

方　法

エンドポイントグループ（EPG）作成は cisco.aci.aci_epg モジュールを利用します。エンドポイントグループのドメインとの関連付けには、cisco.aci.aci_epg_to_domain モジュールを利用します。作成するプレイブックを紹介します（List 8-16）。

List 8-16　エンドポイントグループを作成するプレイブック: ./sdn/aci_epg.yml

```
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local
  vars:
    bd: V1500_BD
    epg: V1500_EPG

  tasks:
    # (1) エンドポイントグループの作成
    - name: create epg
      cisco.aci.aci_epg:
        state: present
        tenant: "{{ tenant }}"
        ap: "{{ ap }}"
        bd: "{{ bd }}"
        epg: "{{ epg }}"
        preferred_group: false
```

```
19:         priority: unspecified
20:
21:     # (2) エンドポイントグループとドメインを関連付け
22:     - name: add epg to domain
23:       cisco.aci.aci_epg_to_domain:
24:         state: present
25:         tenant: "{{ tenant }}"
26:         ap: "{{ ap }}"
27:         epg: "{{ epg }}"
28:         domain: "{{ domain }}"
29:         domain_type: phys
```

解　説

(1) エンドポイントグループの作成

エンドポイントグループ作成の前提条件として、紐付け先のアプリケーションプロファイルが作成されている必要があります。GUI から作る際に必須項目となるブリッジドメインは、モジュールを利用して作成する際には必須項目ではありません。しかし指定がないと値が空のまま作成されるため、ブリッジドメインを用意してからエンドポイントグループを作ることを推奨します。

cisco.aci.aci_epg モジュールの主なパラメータは Table 8-13 のとおりです。

Table 8-13　cisco.aci.aci_epg モジュールの主なパラメータ

パラメータ名	説明
ap	アプリケーションプロファイル名を定義
bd	ブリッジドメイン名を定義
epg	エンドポイントグループ名を定義
preferred_group	プレファードグループに含めるかを指定する（デフォルト: false）
priority	QoS 設定を指定
tenant	テナント名を定義

(2) エンドポイントグループとドメインを関連付け

ドメインと関連付けることにより、VLAN をエンドポイントグループにマッピングできるようになります。

cisco.aci.aci_epg_to_domain モジュールの主なパラメータは Table 8-14 のとおりです。

Table 8-14　cisco.aci.aci_epg_to_domain モジュールの主なパラメータ

パラメータ名	説明
ap	アプリケーションプロファイル名を定義
tenant	テナント名を定義
epg	エンドポイントグループ名を定義
domain	ドメイン名を定義
domain_type	ドメインタイプを指定

関　連

▷ エンドポイントグループを管理する（cisco.aci.aci_epg モジュール）

`https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_epg_module.html`

▷ エンドポイントグループとドメインを関連付ける（cisco.aci.aci_epg_to_domain モジュール）

`https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_epg_to_domain_module.html`

8-3-4　フィルタを作成する

キーワード

▷ cisco.aci.aci_filter モジュール

▷ cisco.aci.aci_filter_entry モジュール

方　法

コントラクトで利用するフィルタを作成するプレイブックを紹介します。cisco.aci.aci_filter モジュールでフィルタを作成します。フィルタにエントリを追加するには、cisco.aci.aci_filter_entry モジュールを利用します（List 8-17）。

List 8-17　フィルタを作成するプレイブック: ./sdn/aci_filter.yml

```
1: ---
2: - hosts: aci
3:   gather_facts: false
4:   connection: ansible.builtin.local
5:   vars:
6:     filter: http_icmp_Filt
```

```
 7:
 8:  tasks:
 9:    # (1) フィルタの作成
10:    - name: create a filter
11:      cisco.aci.aci_filter:
12:        state: present
13:        tenant: "{{ tenant }}"
14:        filter: "{{ filter }}"
15:
16:    # (2) エントリの追加
17:    - name: add a filter icmp entry
18:      cisco.aci.aci_filter_entry:
19:        state: present
20:        tenant: "{{ tenant }}"
21:        filter: "{{ filter }}"
22:        entry: icmp
23:        ether_type: ip
24:        ip_protocol: icmp
25:
26:    - name: add a filter http entry
27:      cisco.aci.aci_filter_entry:
28:        state: present
29:        tenant: "{{ tenant }}"
30:        filter: "{{ filter }}"
31:        entry: http
32:        ether_type: ip
33:        ip_protocol: tcp
34:        dst_port: http
35:
```

解　説

(1) フィルタの作成

cisco.aci.aci_filter モジュールの主なパラメータは Table 8-15 のとおりです。

Table 8-15　cisco.aci.aci_filter モジュールの主なパラメータ

パラメータ名	説明
tenant	テナント名を定義
filter	フィルタ名を定義

(2) エントリの追加

cisco.aci.aci_filter_entry モジュールの主なパラメータは Table 8-16 のとおりです。

Table 8-16　cisco.aci.aci_filter_entry モジュールモジュールの主なパラメータ

パラメータ名	説明
tenant	テナント名を定義
filter	フィルタ名を定義
entry	エントリ名を定義
ether_type	イーサネットタイプを指定（デフォルト: unspecified）
ip_protocol	イーサネットタイプが ip の場合のみ IP プロトコルタイプを指定（デフォルト: unspecified）
dst_port	ip_protocol パラメータが tcp または udp の場合、宛先ポートを定義（デフォルト: unspecified）

利用するパラメータは、作成するフィルタのイーサネットタイプやプロトコルによって異なります。サンプルでは ICMP と HTTP のフィルタを作成しています。ICMP のフィルタでは、entry パラメータを icmp に指定します。プロトコルを icmp にしている場合は icmp_msg_type（ICMP のメッセージタイプ）パラメータなど、プロトコル専用のパラメータが存在します。

HTTP 用のフィルタでは「`dst_port: http`」を指定します。一部アプリケーションはキーワードが用意されているためキーワードで指定できます。「`dst_port: 80`」のようにポート番号で指定しても、プレイブック実行時には冪等性含め同じ動作になります。

関　連

▷ フィルタを管理する（cisco.aci.aci_filter モジュール）

`https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_filter_module.html`

▷ エントリを管理する（cisco.aci.aci_filter_entry モジュール）

`https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_filter_entry_module.html`

8-3-5　コントラクトを作成する

キーワード

▷ cisco.aci.aci_contract モジュール

▷ cisco.aci.aci_contract_subject モジュール

▷ cisco.aci.aci_contract_subject_to_filter モジュール

方 法

コントラクトを作成しフィルタと関連付けするプレイブックを紹介します（List 8-18）。

List 8-18　コントラクトを作成するプレイブック: ./sdn/aci_contract.yml

```
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local
  vars:
    contract: web_http_CT
    subject: filter_subject
    filter: http_icmp_Filt

  tasks:
    # (1) コントラクトの作成
    - name: create a contract
      cisco.aci.aci_contract:
        state: present
        tenant: "{{ tenant }}"
        contract: "{{ contract }}"
        scope: tenant

    # (2) サブジェクトの作成
    - name: create a subject
      cisco.aci.aci_contract_subject:
        state: present
        tenant: "{{ tenant }}"
        contract: "{{ contract }}"
        subject: "{{ subject }}"
        reverse_filter: true
        priority: unspecified

    # (3) サブジェクトとフィルタの関連付け
    - name: add a subject to filter binding
      cisco.aci.aci_contract_subject_to_filter:
        state: present
        tenant: "{{ tenant }}"
        filter: "{{ filter }}"
        contract: "{{ contract }}"
        subject: "{{ subject }}"
        log: log
```

解　説

(1) コントラクトの作成

コントラクトの作成には、cisco.aci.aci_contract モジュールを利用します。

cisco.aci.aci_contract モジュールの主なパラメータは Table 8-17 のとおりです。

Table 8-17　cisco.aci.aci_contract モジュールの主なパラメータ

パラメータ名	説明
tenant	テナント名を定義
contract	コントラクト名を定義
scope	コントラクトのスコープを指定（デフォルト: context）

(2) サブジェクトの作成

コントラクトに関連付けるサブジェクトの作成には、cisco.aci.aci_contract_subject モジュールを利用します。cisco.aci.aci_contract_subject モジュールの主なパラメータは Table 8-18 のとおりです。

Table 8-18　cisco.aci.aci_contract_subject モジュールの主なパラメータ

パラメータ名	説明
tenant	テナント名を定義
contract	コントラクト名を定義
subject	サブジェクト名を定義
reverse_filter	リターン・トラフィックを許可（デフォルト: true）
priority	QoS のクラスを定義（デフォルト: unspecified）

(3) サブジェクトとフィルタの関連付け

cisco.aci.aci_contract_subject_to_filter モジュールの主なパラメータは Table 8-19 のとおりです。

Table 8-19　cisco.aci.aci_contract_subject_to_filter モジュールの主なパラメータ

パラメータ名	説明
tenant	テナント名を定義
filter	フィルタ名を定義
contract	コントラクト名を定義
subject	サブジェクト名を定義
log	ロギングの有効化（デフォルト: none）

ロギングを有効化するには log パラメータに「log」を指定します。

関　連

▷コントラクトを管理する（aci_contract モジュール）

https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_contract_module.html

▷サブジェクトを管理する（aci_contract_subject モジュール）

https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_contract_subject_module.html

▷サブジェクトとフィルタを関連付ける（aci_contract_subject_to_filter モジュール）

https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_contract_subject_to_filter_module.html

8-3-6　エンドポイントグループとコントラクトを関連付ける

キーワード

▷cisco.aci.aci_epg_to_contract モジュール

方　法

作成済みのエンドポイントグループとコントラクトを関連付けるプレイブックを紹介します（List 8-19）。

List 8-19　作成済みのエンドポイントグループとコントラクトを関連付けるプレイブック：./sdn/aci_epg_to_contracts.yml

```
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local
  vars:
    epg: V1500_EPG
    provider_contract: web_http_CT
    consumer_contract: web_http_CT

  tasks:
    # (1) プロバイダコントラクトを設定する
    - name: attached provider contract to epg
      cisco.aci.aci_epg_to_contract:
```

```
14:         state: present
15:         tenant: "{{ tenant }}"
16:         ap: "{{ ap }}"
17:         epg: "{{ epg }}"
18:         contract: "{{ provider_contract }}"
19:         contract_type: provider
20:
21:     # (2) コンシューマコントラクトを設定する
22:     - name: attached consumer contract to epg
23:       cisco.aci.aci_epg_to_contract:
24:         state: present
25:         tenant: "{{ tenant }}"
26:         ap: "{{ ap }}"
27:         epg: "{{ epg }}"
28:         contract: "{{ consumer_contract }}"
29:         contract_type: consumer
```

解　説

エンドポイントグループにコントラクトを関連付けるには、aci_epg_to_contract モジュールを利用します。cisco.aci.aci_epg_to_contract モジュールの主なパラメータは Table 8-20 のとおりです。

Table 8-20　cisco.aci.aci_epg_to_contract モジュールの主なパラメータ

パラメータ名	説明
tenant	テナント名を定義
ap	アプリケーションプロファイル名を定義
epg	エンドポイントグループ名を定義
contract	コントラクト名を定義
contract_type	コントラクトタイプを指定

contract_type パラメータは通信の向きによって provider、consumer を適宜指定します。

関　連

▷ エンドポイントグループとコントラクトを関連付ける（cisco.aci.aci_epg_to_contract モジュール）

https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_epg_to_contract_module.html

8-3-7　エンドポイントグループにポートをアサインする

キーワード

▷ cisco.aci.aci_static_binding_to_epg モジュール

方　法

エンドポイントグループにアクセスポートをアサインするプレイブックを紹介します（List 8-20）。

List 8-20　エンドポイントグループにアクセスポートをアサインするプレイブック:
./sdn/aci_bind_epg_to_static_port.yml

```
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local
  vars:
    epg: V1500_EPG
    leaf: 101
    interfaces:
      - 1/1
      - 1/2
      - 1/3
    vlan_id: 500

  tasks:
    - name: bind epg to vpc_port
      cisco.aci.aci_static_binding_to_epg:
        state: present
        tenant: "{{ tenant }}"
        ap: "{{ ap }}"
        epg: "{{ epg }}"
        encap_id: "{{ vlan_id }}"
        deploy_immediacy: lazy
        interface_mode: tagged
        interface_type: switch_port
        pod_id: 1
        leafs: "{{ leaf }}"
        interface: "{{ item }}"
      loop: "{{ interfaces }}"
```

解　説

エンドポイントグループにアクセスポートをアサインするには、cisco.aci.aci_static_binding_to_epg モ

ジュールを利用します。cisco.aci.aci_static_binding_to_epg モジュールの主なパラメータは Table 8-21 のとおりです。

Table 8-21　cisco.aci.aci_static_binding_to_epg モジュールの主なパラメータ

パラメータ名	説明
tenant	テナント名を定義する
ap	アプリケーションプロファイル名を定義する
epg	エンドポイントグループ名を定義する
encap_id	関連付ける VLAN ID を定義する
deploy_immediacy	デプロイのタイミングを指定する（デフォルト: lazy）
interface_mode	インターフェイスモードを指定する（デフォルト: trunk）
interface_type	インターフェイスタイプを指定する（デフォルト: switch_port）
pod_id	ポッド ID を定義する
leafs	リーフを定義する
interface	インターフェイスを定義する

リーフ、インターフェイスを定義して 1 ポートずつアサインするモジュールです。複数のポートにアサインが必要な場合は loop ディレクティブを活用するのが効果的です。interface_mode パラメータは選択肢が以下の 7 個用意されています。

- 802.1p
- access
- native
- regular
- tagged
- trunk
- untagged

802.1p と native、access と untagged、trunk と tagged と regular はそれぞれ同じ動作をします。

応　用

応用として、VPC の設定がされたポートにエンドポイントグループをアサインするプレイブックを紹介します（List 8-21）。

List 8-21　エンドポイントグループに VPC ポートをアサインするプレイブック: ./sdn/aci_bind_epg_to_static_port_vpc.yml

```
1: ---
2: - hosts: aci
3:   gather_facts: false
```

```
 4:  connection: ansible.builtin.local
 5:  vars:
 6:    epg: Vl500_EPG
 7:    leaf: 101-102
 8:    interface: Sample_VPC
 9:    vlan_id: 500
10:
11:  tasks:
12:    - name: bind epg to static_port
13:      cisco.aci.aci_static_binding_to_epg:
14:        state: present
15:        tenant: "{{ tenant }}"
16:        ap: "{{ ap }}"
17:        epg: "{{ epg }}"
18:        encap_id: "{{ vlan_id }}"
19:        deploy_immediacy: lazy
20:        interface_mode: native
21:        interface_type: vpc
22:        pod_id: 1
23:        leafs: "{{ leaf }}"
24:        interface: "{{ interface }}"
```

interface_type パラメータと interface パラメータ、leafs パラメータの指定方法が変わります。interface_type パラメータは vpc、interface は VPC のインターフェイスプロファイル名、leafs はインターフェイスプロファイルで登録したリーフをそれぞれ定義します。2 台のリーフにまたがる設定のため、leafs パラメータは「101-102」のように「-」で繋いだリーフ名を定義します。

関　連

▷エンドポイントグループにポートにアサインする（cisco.aci.aci_static_binding_to_epg モジュール）
https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_static_binding_to_epg_module.html

8-4　運用管理

Cisco ACI における運用管理を行うためのプレイブックを紹介します。スナップショットの作成やユーザ作成など運用する上で不可欠な作業を自動化します。

8-4-1　スナップショット作成する

キーワード

▷ cisco.aci.aci_config_snapshot モジュール

方　法

システム構成のバックアップである、スナップショットを作成するプレイブックを紹介します（List 8-22）。

List 8-22　スナップショットを作成するプレイブック: ./sdn/aci_snapshot.yml

```
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local

  tasks:
    - name: create a latest snapshot
      cisco.aci.aci_config_snapshot:
        state: present
        export_policy: APIC
```

解　説

スナップショットの作成は cisco.aci.aci_config_snapshot モジュールを使用します。

cisco.aci.aci_config_snapshot モジュールの主なパラメータは Table 8-22 のとおりです。

Table 8-22　cisco.aci.aci_config_snapshot モジュールの主なパラメータ

パラメータ名	説明
export_policy	エクスポートポリシー名を定義

export_policy パラメータの値とタスク実行時の日時を組み合わせて自動的にスナップショットのファイルが APIC 上に生成されます。ファイル生成時の命名規則は「ce_<export_policy>-<yyyy>-<mm>-<dd>T<hh>-<mm>-<ss>」です（Operation 8-5）。

Operation 8-5　生成されるスナップショットファイル（2021/8/20 12:00 に実施）

```
ce_APIC-2021-8-20T12-00-00.tar.gz
```

また、存在しないエクスポートポリシーを指定した場合、実行時に合わせて作成されます。

応　用

応用としてファイルの世代管理方法について紹介します。max_count パラメータを利用することでエクスポートポリシーごとに指定数のファイル（サンプルでは 2 個）を保持できます。指定できる値は 1～10 です。プレイブック実行時、スナップショットの保存数が max_count パラメータで指定した数を超えている場合は、古いスナップショットから削除されるので注意してください（List 8-23）。

List 8-23　スナップショットを 2 世代まで管理するプレイブック: ./sdn/aci_snapshot_advanced.yml

```
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local

  tasks:
    - name: create a latest snapshot
      cisco.aci.aci_config_snapshot:
        state: present
        export_policy: APIC
        max_count: 2
```

関　連

▷ スナップショットを取得する（cisco.aci.aci_config_snapshot モジュール）

https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_config_snapshot_module.html

▷ スナップショットを利用したロールバックをする（cisco.aci.aci_config_rollback モジュール）

https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_config_rollback_module.html

8-4-2　ユーザを作成する

キーワード

▷ cisco.aci.aci_user モジュール

方　法

ユーザ作成をするプレイブックを紹介します。事前に pip で python-dateutil のインストールが必要です（List 8-24）。

List 8-24　ユーザを作成するプレイブック: ./sdn/aci_user.yml

```
{
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local
  vars:
    aaa_user: TestUser001
    aaa_password: P@ssw0rd!!!

  tasks:
    - name: add a user
      cisco.aci.aci_aaa_user:
        state: present
        aaa_user: "{{ aaa_user }}"
        aaa_password: "{{ aaa_password }}"
        aaa_password_update_required: false
        expiration: never
        expires: false

```

解　説

ユーザの作成には cisco.aci.aci_aaa_user モジュールを利用します（Table 8-23）。

Table 8-23　cisco.aci.aci_aaa_user モジュールの主なパラメータ

パラメータ名	説明
aaa_user	アカウント名を定義
aaa_password	アカウントのパスワードを定義
aaa_password_update_required	パスワードの更新が必要、不要を指定
expiration	アカウントの有効期限を定義
expires	アカウントの有効期限を有効化、無効化を指定

本モジュールは冪等性が担保できません。プレイブックを再度実行すると、パスワードの更新にな

るため、同一のパスワードは設定できないというエラーメッセージが表示されます（Operation 8-6）[*4]。

Operation 8-6　サンプルプレイブックを再実行した際のエラー

```
...（ 略 ）...
TASK [add a user] ********************************************************************************
[WARNING]: Module did not set no_log for aaa_password_update_required
[WARNING]: Module did not set no_log for aaa_password_lifetime
[WARNING]: Module did not set no_log for clear_password_history
fatal: [apic]: FAILED! => {"changed": false, "error": {"code": "1", "text": "Password hist
ory check: user TestUser001 should not use previous 5 passwords"}, "msg": "APIC Error 1: P
assword history check: user TestUser001 should not use previous 5 passwords"}
...（ 略 ）...
```

WARNING の内容は、一部パラメータ名に password という文字列が含まれているため、Ansible 自体が表示しています。処理には影響がないため無視しても問題ありません。

本モジュールは、ユーザの作成のみができます。セキュリティドメインとの関連付けはできません。セキュリティドメインとの関連付けをするモジュールがないため、cisco.aci.aci_rest モジュールを利用して設定ください。

応　用

応用として、cisco.aci.aci_rest モジュールを利用したセキュリティドメインとの関連付けについて紹介します。ユーザの作成時には読み取り権限しか付与されないため、admin ロールを関連付けます（List 8-25）。

List 8-25　ユーザ作成後にセキュリティドメインとの関連付けをするプレイブック:
./sdn/aci_user_advanced.yml

```
1: ---
2: - hosts: aci
3:   gather_facts: false
4:   connection: ansible.builtin.local
5:   vars:
6:     aaa_user: TestUser001
7:     aaa_password: P@ssw0rd!!!
8:
9:   tasks:
10:    - name: add a user
11:      cisco.aci.aci_aaa_user:
```

＊4　直近 5 回で使用した過去のパスワードと同じものは設定できない ACI の仕様のため。

```
        state: present
        aaa_user: "{{ aaa_user }}"
        aaa_password: "{{ aaa_password }}"
        aaa_password_update_required: false
        expiration: never
        expires: false

    - name: attach admin role
      cisco.aci.aci_rest:
        method: post
        path: "api/node/mo/uni/userext/user-{{ aaa_user }}/userdomain-all.json"
        content:
          aaaUserDomain:
            attributes:
              dn: "uni/userext/user-{{ aaa_user }}/userdomain-all"
              name: "all"
              rn: "userdomain-all"
            children:
              - aaaUserRole:
                  attributes:
                    dn: "uni/userext/user-{{ aaa_user }}/userdomain-all/role-admin"
                    name: "admin"
                    privType: "writePriv"
                    rn: "role-admin"
```

cisco.aci.aci_rest モジュールを利用することでセキュリティドメインの関連付けができました。cisco.aci.aci_rest モジュールの解説については 8-4-3 を参照してください。

関　連

▷ ユーザを管理する（cisco.aci.aci_aaa_user モジュール）

https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_aaa_user_module.html

▷ 8-4-3 モジュールがない作業も Ansible で自動化する（cisco.aci.aci_rest モジュール）

8-4-3　モジュールのない作業を Ansible で設定する

キーワード

▷ cisco.aci.aci_rest モジュール

方　法

専用のモジュールのない作業を Ansible で設定できます。サンプルではインターフェイスを有効化するプレイブックを紹介します（List 8-26）。

List 8-26　インターフェイスを有効化するプレイブック: ./sdn/aci_rest_post.yml

```
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local
  vars:
    pod_id: 1
    leaf: 101
    eth:
      - 1/21
      - 1/22

  tasks:
    - name: interface no shutdown
      cisco.aci.aci_rest:
        method: post
        path: "/api/node/mo/uni/fabric/outofsvc.json"
        content:
          fabricRsOosPath:
            attributes:
              dn: "uni/fabric/outofsvc/rsoosPath-[topology/pod-{{ pod_id }}/⇒
  paths-{{ leaf }}/pathep-[eth{{ item }}]]"
              status : "deleted"
      loop: "{{ eth }}"
```

解　説

モジュールがない作業を自動化するには、cisco.aci.aci_rest モジュールを利用します。cisco.aci.aci_rest モジュールを使うには、APIC REST API についての知識が必要です。cisco.aci.aci_rest モジュールの主なパラメータは Table 8-24 のとおりです。

Table 8-24　cisco.aci.aci_rest モジュールの主なパラメータ

パラメータ名	説明
method	HTTP メソッドを指定する
path	リクエスト URI を定義する
content	API リクエストのペイロードを定義する

path パラメータは「/api/」以降のパスを指定します。パスは「.json」、または「.xml」で終わります。content パラメータの定義方法は以下の 3 種類あります。サンプルのプレイブックでは可読性の高い YAML 形式で定義しています。

- YAML 形式で定義
- JSON 形式で定義
- XML 形式で定義

XML 形式を利用する場合、lxml、xmljson ライブラリを pip でインストールする必要があります。

応　用

cisco.aci.aci_rest モジュールの応用方法として、フォルトの取得をするプレイブックを紹介します。運用上確認することが多いフォルトですが、専用のモジュールはありません。そこで cisco.aci.aci_rest モジュールと ansible.buildin.template モジュールを組み合わせて CSV ファイルに出力します（List 8-27、List 8-28）。

List 8-27　テンプレートファイル: ./sdn/templates/fault_to_csv.j2

```
"severity","domain","code","rule","descr"
{% for r in res_faults['imdata'] %}
{% set faultInst = r['faultInst']['attributes'] %}
"{{ faultInst['severity'] }}","{{ faultInst['domain'] }}","{{ faultInst['code'] }}",⇒
  "{{ faultInst['rule'] }}","{{ faultInst['descr'] }}"
{% endfor %}
```

List 8-28　フォルトを取得し CSV ファイルに出力する: ./sdn/aci_rest_get.yml

```
---
- hosts: aci
  gather_facts: false
  connection: ansible.builtin.local

  tasks:
    - name: query faults
      cisco.aci.aci_rest:
        path: /api/node/class/faultInst.json
        method: get
      register: res_faults

```

```
13:     - name: output csv file
14:       ansible.builtin.template:
15:         src: faults_to_csv.j2
16:         dest: files/faults_result.csv
17:
```

サンプルでは、すべてのフォルトを取得するプレイブックになっています。取得する情報量が多くなってしまう場合は、パスにオプションを追加して、情報を絞って取得する方法もあります。

関　連

▷モジュールがない作業を Ansible で設定する（cisco.aci.aci_rest モジュール）

https://docs.ansible.com/ansible/latest/collections/cisco/aci/aci_rest_module.html

Column　APIC REST API のパスを調べるには

APIC には、API を調べるのに便利な API Inspector という機能があります。APIC ログイン後に画面右上にある歯車（Help and tools）ボタンから起動できます。起動すると別のプロンプトが立ち上がり、GUI で行う作業の Method、URL、Response が表示されます。この機能と cisco.aci.aci_rest モジュールを活用することでほぼすべての作業を Ansible で実現できます。

https://learningnetwork.cisco.com/s/article/api-inspector-x

終わりに

本書をお読みいただき、誠にありがとうございます。本書は Ansible の入門書の次の 1 冊として執筆しました。このコンセプトに至った背景を説明します。

私は普段、Ansible を中心とした自動化支援業務のほか、社内外の自動化トレーニングの講師や、Ansible ユーザー会などのコミュニティへの参加をしています。Ansible を学習する方々と接する中で共通の悩みを耳にしてきました。「基礎は分かったが、実現したいことに対してどのモジュールを使ってどうプレイブックを書けばいいか分からない」といった悩みです。私自身もモジュールをどう組み合わせたらよいかなど、似た悩みを持っていた経験がありました。

Ansible は対応するプラットフォームの範囲が広く、多数のモジュールがあります。せっかく便利なツールなのに、実現したいことと機能がうまく結びついていない状態は、とてももったいなく感じました。そこで本書では、Ansible の機能ではなく「実現したいこと」を切り口にしました。これにより、実現したいことに対してプレイブックをどう書けばよいかを導けるようにしました。

本書の性質上、必要な箇所を読み終えたらいったん本を閉じることでしょう。繰り返しになりますが、Ansible の対応範囲は広いです。また Ansible で実現したいことができたときに、ぜひ本書を思い出して開いていただければ幸いです。

本書が、Ansible をより活用するきっかけになることを願っています。

謝辞

本書の執筆、出版にあったては多くの方々のご協力を賜りました。この場をお借りして感謝申し上げます。

土屋様には執筆の機会をいただき、度々のスケジュール調整にもご対応いただきました。出版まで併走していただき誠にありがとうございました。

共著者である大嶋健容様、三枝浩太様、宮崎啓史様にはお忙しい中で執筆にご協力いただきました。私一人では Ansible の幅広い対応範囲を決してカバーできません。みなさまのプロフェッショナルとしての知識と経験があったからこそ、本書ができました。

佐藤学様には本書の構想段階で相談に乗っていただきました。おかげさまで具体的に構成をイメージでき、執筆を始めることができました。Ansible 実践ガイドの初版からの著者である北山晋吾様には、図解に利用する素材をご提供いただき、統一感のある図を作成できました。

私が所属する株式会社エービーコミュニケーションズの関係者には、業務時間の確保や検証環境などの配慮をいただきました。執筆活動に理解をいただき大変助かりました。また、私事ですが家族にも支えられました。ときどき息抜きをすすめてくれたおかげで長丁場をやりきれました。

改めて、周りの方々のありがたさを感じる機会となりました。本当にありがとうございました。

2022 年 2 月
横地 晃

索引

B

C

D

E

F

G

H

I

J

K

L

● 著者プロフィール

■ 大嶋 健容（おおしま たけひろ）

Linux を使用した B2B 向けインターネットシステム（Web、Mail、Proxy サーバなど）の設計・構築・運用を経験した後に自社サービスの新規クラウド事業立ち上げプロジェクトに参画。プロジェクトではエンドユーザへ提供する仮想サーバやネットワーク、ストレージなどのリソース構築および運用を効率化するための自動化を担当し Ansible と出会う。自動化の実装を進めていく中で Ansible の便利さに気づきアップストリームのモジュール開発活動や日本のコミュニティである「Ansible ユーザー会」への貢献を始める。現在はレッドハット株式会社にて Ansible を中心とした自動化のコンサルタントとして勤務している。

■ 三枝 浩太 (さえぐさ こうた)

官公庁向けネットワークの設計、構築業務を経験。2017 年に株式会社エーピーコミュニケーションズへ入社。入社後 Python を利用したネットワーク運用自動化に興味を持ち学習をしていた際に Ansible と出会う。その後、Cisco ACI や VMware NSX における設計・運用支援などに携わったことで、SDN と Ansible の相性の良さに気づき、Ansible を用いた運用自動化を担当する。現在はエンドユーザ向けに Ansible を中心としたネットワーク自動化導入支援プロジェクトに従事している。趣味は Nike のスニーカー蒐集。

■ 宮崎 啓史（みやざき ひろし）

ISP 向けの大規模メールサービスや web のインフラ構築およびバックエンド開発、業務向けのフィーチャーフォンアプリや組み込み開発を経験。その後、業務を通して Ansible やコンテナを使ったインフラのコード化に興味を持ち、2021 年に株式会社エーピーコミュニケーションズへ入社。現在は Ansible を使ったネットワーク自動化を推進する部署に所属。プライベートでは Ansible やクラウドネイティブなどのインフラ技術関連のコミュニティへ積極的に参加している。「業務を効率化し、いかに少ない労力で済ませるか」を考えることに労力を費やすのが好き。朝晩の散歩が日課。

■ 横地 晃（よこち あきら）

インフラ構築からシステム開発まで行う SIer にてエンジニアキャリアをスタートし、さまざまな業務を経験。その後、株式会社エーピーコミュニケーションズにて、データセンタやエンタープライズ向けネットワークの設計構築業務に従事。自動化に関心を持ち始めた頃に、ネットワーク機器に対応した Ansible に出会う。現在は、Ansible を中心としたネットワーク運用自動化の支援業務や自動化トレーニングの講師を担当している。

その他、「Ansible ユーザー会」などのコミュニティへの参加や運営、技術ブログの投稿など、アウトプットすることがライフワーク。

社内では、エンジニアが自律的にアウトプットや学習をするための支援をするメンターも担当しており、人材育成にも力を入れている。

● お断り

IT の環境は変化が激しく、Ansible の展開する自動化の分野もアップデートが頻繁に行われます。本書に記載されている内容は、2022 年 2 月時点のものですが、サービスの改善や新機能の追加は、日々行われているため、本書の内容と異なる場合があることは、ご了承ください。また、本書の実行手順や結果については、筆者の使用するハードウェアとソフトウェア環境において検証した結果ですが、ハードウェア環境やソフトウェアの事前のセットアップ状況によって、本書の内容と異なる場合があります。この点についても、ご了解いただきますよう、お願いいたします。

● 正誤表

インプレスの書籍紹介ページ https://book.impress.co.jp/books/1120101163 からたどれる「正誤表」をご確認ください。これまでに判明した正誤があれば「お問い合わせ／正誤表」タブのページに正誤表が表示されます。

● スタッフ

AD ／装丁：岡田 章志＋ GY

本文デザイン／制作／編集：TSUC LLC

■商品に関する問い合わせ先

このたびは弊社商品をご購入いただきありがとうございます。本書の内容などに関するお問い合わせは、下記のURLまたはQRコードにある問い合わせフォームからお送りください。

https://book.impress.co.jp/info/

上記フォームがご利用頂けない場合のメールでの問い合わせ先

info@impress.co.jp

※お問い合わせの際は、書名、ISBN、お名前、お電話番号、メールアドレス に加えて、「該当するページ」と「具体的なご質問内容」「お使いの動作環境」を必ずご明記ください。なお、本書の範囲を超えるご質問にはお答えできないのでご了承ください。

●電話やFAX でのご質問には対応しておりません。また、封書でのお問い合わせは回答までに日数をいただく場合があります。あらかじめご了承ください。
●インプレスブックスの本書情報ページ https://book.impress.co.jp/books/1120101163 では、本書のサポート情報や正誤表・訂正情報などを提供しています。あわせてご確認ください。
●本書の奥付に記載されている初版発行日から3 年が経過した場合、もしくは本書で紹介している製品やサービスについて提供会社によるサポートが終了した場合はご質問にお答えできない場合があります。

■落丁・乱丁本などの問い合わせ先

TEL 03-6837-5016 FAX 03-6837-5023

service@impress.co.jp

（受付時間／10:00〜12:00、13:00〜17:30土日祝祭日を除く）

※古書店で購入された商品はお取り替えできません。

■書店／販売会社からのご注文窓口

株式会社インプレス 受注センター

TEL 048-449-8040

FAX 048-449-8041

アンシブル

Ansible クックブック

2022年 3月11日 初版第1刷発行

著 者 大嶋 健容（おおしま たけひろ）、三枝 浩太（さえぐさ こうた）、宮﨑 啓史（みやざき ひろし）、横地 晃（よこち あきら）

発行人 小川 亨

編集人 高橋隆志

発行所 株式会社インプレス
〒101-0051 東京都千代田区神田神保町一丁目105番地
ホームページ https://book.impress.co.jp/

印刷所 大日本印刷株式会社

ISBN978-4-295-01354-9 C3055

Printed in Japan